Software Development for the Detection of $GABA_A$ Mediated Synaptic Currents and Its Application in Rat Substantia Nigra Neurons

by: Wu Cheok Wai

A thesis submitted as a partial fulfillment of

the requirement for the degree of

Master of Philosophy in Basic Medical Sciences

June 1998

Department of Physiology

Division of Basic Medical Sciences

The Chinese University of Hong Kong


香港中文大學
圖書館系統
15 JUL 1999
UNIVERSITY
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

Table of Contents

Abstract

GABA is the major inhibitory neurotransmitter in the central nervous system. The ubiquitous GABAergic interneurons provide a powerful inhibitory control of the general neuronal excitability while specific GABAergic projection neurons serve various functions in different brain circuits. The present thesis describes the development of a software for automatic detection and characterization of $GABA_A$ receptor mediated inhibitory postsynaptic currents (IPSCs) recorded by the voltage-clamp technique.

The algorithm used is intuitive and is essentially a multi-criteria test. First, distinct features derived from the shapes of typical $GABA_A$ IPSCS were obtained. Based on these features, three specific tests concerning the threshold, rising slope and decay slope were devised. In addition, a fast noise test was implemented. A candidate event will be considered a real event only if it passes all these criteria. In addition, the stringency of the criteria is user-adjustable to make the program more flexible. The software also measures parameters such as amplitude, 20-80% rise time and decay time constants of the IPSCs. The performance of the software was evaluated by an objective method. The consistency between the IPSCs detected by the software and a human subject is 88.7%. The value is comparable with those obtained by two different subjects.

In an application example, the software was used to characterize the spontaneous IPSCs of the dopaminergic neurons of substantia nigra pars compacta. The mean amplitude, 20-80% rise time and decay time constant were

59.68±4.15 pA, 1.04±0.06 ms and 10.67±0.51 ms respectively (38 cells). The program also revealed that the novel GABA uptake inhibitor, NO-711 Hydrochloride, significantly increased the decay time constants of the spontaneous IPSCs suggesting a role of GABA uptake in terminating the action of GABA on these neurons.

摘要

本論文描述一軟件的製作，用以自動偵測γ氨基丁酸A受體引發的抑制性突觸後電流，並詳細描述其中所採用的算法。這軟件的性能測試結果亦會公報。最後，這軟件被用來偵測黑質多巴胺能神經元的突觸電流，以作為一應用例子。

γ氨基丁酸是腦中一主要的抑制性神經遞質。γ氨基丁酸中間神經元強烈地抑制其他神經元的興奮性。本論文主要介紹其中軟件的制作，用以自動偵測γ氨基丁酸A受體引發的突觸電流。

這軟件的算法是因應抑制性突觸後電流的特性而設計的，它仔細地檢查抑制性突觸後電流的幅度、尖、上升期及衰變期。這軟件並量度所偵測到的抑制性突觸後電流的20-80%上升期及衰變常數。這軟件有人類一樣的表現，所偵測到的抑制性突觸後電流有88.7%和所人類偵測到的相同。這數字和人類之間的相同百分比相約。

在這軟件的一應用例子中，它用來偵測多巴胺能神經元的自發抑制性突觸後電流。所量度到的平均幅

度、20-80%上升期及衰變常數分別為59.68±4.15 pA、1.04±0.06 ms及10.67±0.51 ms（38個細胞）。NO-711 Hydrochloride是一創新的γ氨基丁酸的攝取抑制因子，它明顯地增加抑制性突觸後電流的衰變常數並減少其頻率。

Acknowledgment

I would like to thank my supervisor, Dr. W.H. Yung, for his advice on writing this thesis and his support of the research project. Through the discussions with him, many new ideas have been produced. I also want to thank Mr. Kenny Ho for his help on computer hardware maintenance. Thanks are also given to Dr. M. S. Leung for his advice on the skills of using computers. I also want to thank all other people who have given me supports, encouragement and guidance.

Chapter 1: Introduction

The primary purpose of the present project is to develop a program that facilitates the automation of the detection of fast synaptic currents, especially those mediated by $GABA_A$ receptors. In this introductory chapter, the process of GABA neurotransmission, its importance in the central nervous system, and characteristics of $GABA_A$ mediated inhibitory postsynaptic currents (IPSCs) are briefly described. This is followed by a discussion of some of the approaches, and their limitations, most commonly employed to detect IPSCs, and the need to develop our own software. Finally, some of the features of the C++ language, which is the programming tool used to develop the software, are described.

1.1 $GABA_A$ Mediated Neurotransmission

Gamma-aminobutyric acid (GABA) is the major inhibitory neurotransmitter in the brain (Srinivasan *et al.*, 1969; Fonnum and Walberg, 1973; Rabow *et al.*, 1995) . It is synthesized from glutamic acid by the enzyme glutamic acid decarboxylayse in a single step. There are two major types of GABA receptor: $GABA_A$ and $GABA_B$. $GABA_A$ receptors directly link to Cl^- channels, forming a receptor-channel complex (Rabow *et al.*, 1995). Binding of GABA to $GABA_A$ receptors opens the Cl^- channel leading to an increased conductance to Cl^-. The most common consequence is the influx of the Cl^- into the neuron leading to an inhibitory synaptic current in the postsynaptic neuron.

$GABA_A$ receptors are sensitive to bicuculline, which blocks the binding of GABA in a competitive manner (Curtis *et al.*, 1971; Llano and Gerschenfeld, 1993) and to picrotoxin, which directly blocks the Cl^- channel (Olsen, 1981). On the other hand, $GABA_B$ receptors are coupled to K^+ channels via G-protein (Hill and Bowery, 1981; Andrade *et al.*, 1986). Increased K^+ conductance would lead to the appearance of inhibitory postsynaptic currents. However, in contrary to $GABA_A$ mediated IPSCs, $GABA_B$ IPSCs have a much slower kinetics (Hausser and Yung, 1994). Current wisdom is that $GABA_A$ receptors are the major receptor subtype that mediate the inhibitory effect of GABA. Although $GABA_B$ receptors may also be important, our understanding of its role in the process of inhibitory neurotransmission is still lagging behind. Specific antagonists also exist for $GABA_B$ receptors. For example, activation of $GABA_B$ receptors can be blocked by 2-hydroxysaclofen (Fromm *et al.*, 1992; Solis and Nicoll, 1992).

The $GABA_A$ receptors are implicated in various neurologic and psychiatric diseases, such as anxiety, sleep disorders and epilepsy. They are also the site of action of many clinically useful drugs. For example, benzodiazepines, drugs used to reduce anxiety, bind to the $GABA_A$ receptors and enhance the effect of GABA. They increase the $GABA_A$ mediated synaptic currents by increasing the opening frequency of the $GABA_A$ receptor-channel (Study and Barker, 1981; Vicini *et al.*, 1987). Barbiturates, a class of drugs used as sedatives and to treat epilepsy, increase the average open duration of GABA-gated Cl^--channels (Barker and McBurney, 1979; Study and Barker, 1981; Macdonald and Olsen, 1994). These and other manipulations resulting in

prolongation of IPSCs decay, or increased GABA concentration in the synaptic clefts, are therefore useful targets in drug design.

It is believed that the variety of actions of drugs on the $GABA_A$ receptor are due to the importance of GABA neurones in the brain. First of all, GABA interneurons are ubiquitous in the brain and they provide a powerful inhibitory control of the excitability of the neurons. Spontaneous release of GABA from the terminals of the inhibitory interneurons produces tonic inhibition of neurons in the central nervous system. Thus, blockade of the $GABA_A$ mediated IPSCs by picrotoxin or bicuculline will increase the excitability of the neurons while facilitation will have opposite effects (Dingledine and Gjerstad, 1980). In fact, it has been demonstrated that a slight change of the GABAergic inhibition can result in a far greater change in the activity of the postsynaptic neurons (Stelzer 1992). Furthermore, GABAergic projections from one brain nucleus to another also play important roles in different functional circuits. To take one example, the basal ganglia, an important circuit for control of movement, consists of a set of richly interconnected subcortical nuclei many of which contain GABA projection neurones (Tepper *at el.*, 1995). Inhibition could be achieved by a direct GABA innervation.

1.2 Determinants of $GABA_A$ IPSC Kinetics

Like other fast synaptic currents, $GABA_A$ mediated IPSCs are characterised by a fast onset and a relatively long decay. It is believed that the time course of the rising phase reflects the rate of activation of the Cl^- channels (Bier *et al.*, 1996). There is no consensus in the literature concerning whether

the decay phase could be well described by a single exponential, double exponential or other forms of decay. This fact probably reflects the differences in the characteristics among individual GABA synapses and the variety of mechanisms involved in the termination of the GABA action. As there is no extra-cellular enzyme to break down the GABA after they are released, these mechanisms may include the rate of receptor inactivation, desensitization, diffusion of GABA from the synaptic cleft and the rate of active uptake by specific GABA transporter (Schousboe, 1981; Borowsky and Hoffman, 1995).

There is no easy way to dissect out the contributions of all these factors in determining the duration of GABA action. Nevertheless, the contribution of the uptake process can be determined relatively easily by the use of GABA uptake blockers such as nipecotic acid and its analogues. For example, tiagabine, a GABA uptake blocker used to treat epilepsy, has been used to study the role of GABA uptake on the kinetics of the IPSCs. In the hippocampal slice preparation of the rat, tiagabine causes a large increase in the half-width of the stimulated IPSCs recorded from CA1 neurons (Roepstorff and Lambert, 1992) suggesting a role of uptake in terminating GABA action. However, NO-711 hydrochloride, another analogue of nipecotic acid (Suzdak *at el.*, 1992), did not prolong the decay time of spontaneous IPSCs in the cultured hippocampal neurons (Oh and Dichter, 1994). The disparity could be due to differences in the exact synaptic geometry, or differences in the environment of the synapses in brain slice and culture. Nevertheless, these data point out that the result from one preparation may not be generalised to other brain area or preparation.

1.3 Detection of the Spontaneous IPSCs

As most spontaneous IPSCs exhibit a broad range of amplitude, rise time and decay time, a large number of IPSCs are usually needed to be analyzed in order to fully describe their behavior. For a long time, detection and measurement of spontaneous electrophysiological events were performed manually (Del Castillo and Katz, 1954). This is obviously a very laborious and time consuming method. However, in the past two decades, as the desktop computing power increased tremendously accompanied with big drops in prices, electrophysiologists from different laboratories began to develop their own softwares to specifically detect spontaneous synaptic potentials and currents.

There is a variety of approaches in the detection of spontaneous synaptic events. Nevertheless, the algorithms involved can be roughly divided into two categories: the threshold amplitude method and the scaled template method. In the first approach, a signal is considered to be a synaptic event if its amplitude exceeds a detection threshold (Otis and Mody, 1992; Salin and Prince, 1996). As the rising slope is proportional to the amplitude, one variant of this method considers the rising slope instead of the amplitude. A signal is considered to be an event if the its rising slope exceeds a threshold value (Cochran, 1993; Ankri *et al.*, 1994; Franaszczuk *et al.*, 1995). The advantage is that these methods are relatively simple and easy to implement. However, they are relatively non-selective. Any noise or artifact transient that is large enough will also be considered to be an event, regardless of its shape. One approach to solve this

problem is to allow the detected events to be screened visually and the artifacts are rejected manually.

The scaled template method is more recently introduced (Clements and Bekkers, 1997). It employs a template function to fit the data. The template function usually has a flat baseline region followed by an idealized synaptic time course consisting of an exponential rise and decay. A typical template function is shown below:

$$\text{TEMPLATE}(t) = 0 \quad (t \leq 0)$$

$$\text{TEMPLATE}(t) = \text{NORM}(1-\exp(-t/\text{RISE}))\exp(-t/\text{DECAY}) \quad (t > 0)$$

whereas TEMPLATE(t) represents the idealized synaptic current, t is the time from onset of the idealized synaptic event, NORM is the scaling factor used to normalize the peak amplitude, RISE is the time constant of the rising phase, and DECAY is the time constant of the falling phase. A signal is considered to be a synaptic event if it can be fitted with the above equation with a standard error less than a specific value. The advantage of this method is that it considers the shape of the signal. Fast transient noise will not match the template and will be rejected. However, as the template function predefines the shape of the events including the decay kinetics, if an event has decay time that differs largely from the predefined value, the sensitivity of the program will drop sharply. Furthermore, when two events overlap together, which are not uncommon, they will also produce a shape that do not match the template. The standard error between the signals and the template function increases accordingly. One or

both events may escape from the detection. In other words, this type of algorithm suffers the problem of being over-rigid.

There are other practical limitations when attempting to use software developed by others. First of all, many of these programs, which may be available on request, are oriented to detect synaptic potentials instead of the synaptic currents (Otis and Mody, 1992; Cochran, 1993; Ankri *et al.*, 1994; Franaszczuk *et al.*, 1995; Salin and Prince, 1996). Secondly, most of these programs, even if they are applied to IPSCs, can only read data in a certain format. However, different laboratories may generate data in different formats, which arise as a result of using different analog-digital interfaces and acquisition software. In our laboratory, for example, we make use of the Patch and Voltage Clamp software package and CED1401+ interface provided by Cambridge Electronics Design (Cambridge, U.K.) to generate the IPSCs data, which are incompatible to those generated by other systems, such as those provided by Axon Instruments (CA, USA). Thus, in view of this and the limitations of the detection algorithms invented by others, we found us to have the practical need to develop our own software. The aim of the present thesis is therefore to develop a program which abstract useful features of the inhibitory synaptic currents for their accurate detection. The performance of the program will be fully evaluated and then applied to a real biological GABA synapse with the aim that a minimum manual intervention is needed.

1.4 The C++ Programming Language

We aim to develop a program that can detect spontaneous IPSCs accurately and yet easy to use. After some careful considerations, we decided to adopt the latest version (5.0) of Microsoft Visual C++ language as the development tool. The general characteristics of the C and related languages, and therefore the justification of using Microsoft Visual C++, are outlined below.

The C and C++ Programming Language

C is a general purpose programming language. It was designed by Dennis Ritchie of Bell Laboratory in 1970s. It is the most powerful programming language among those commonly used ones, such as BASIC, FORTRAN and Pascal. It combines together the advantages of both low level and high level computer languages. On one hand, it has the functionality of low level assembly language like directly manipulating bits, bytes and addresses. Pointer, a data type that can store the addresses of variables, is one of the most powerful features of the C language. On the other hand, like other high level computer languages, it can also be used to write structured programs. By using separated small functions to complete each task, very complex programs can be written. Besides, the C language is very rich and versatile. It provides many operators for programming, and library functions for direct manipulation of binary files.

C is a standardized language. The code written in C is highly portable and can be easily implemented in different computer platforms. Although the exact speed of execution of a C program depends on factors like the speed of the main processor and the compiler used, the executable files generated after

compilation usually run very fast. C is the most common language used at present for professional programming projects. Many commercial programs and operating systems are written in the C language (Schildt, 1990; Dempster, 1993).

C++ is a super set of the C language. In addition to the traditional features of the C language, it supports object-orientated programming. It has many useful properties such as inheritance, data encapsulation and data abstraction. These features simplify the task of writing very complex and large programs (Andrews, 1996).

Microsoft Visual C++

As a graphic user interface is much easier to use than a text based interface, many software's operating systems have changed from MS-DOS® to Windows® after the introduction of the latter in PC computers. However, historically, writing Windows based programs is extremely difficult because the codes for graphic user interfaces are usually very complex. Many codes are needed to develop the interface and take care of the Window messages, which convey the users' messages to the programs and vice versa.

Microsoft Visual C++® is one of the compilers commercially available that makes the task of writing Window based programs easier. It provides numerous tools that can be used to take care of details of Windows programming. Its Microsoft Foundation Class (MFC) library contains functions that can be used to create windows, dialog boxes, device contexts and controls. These features greatly reduce the time needed to write Windows based

software. Indeed, Visual C++ is widely used to write professional Windows based software. Version 5 of the compiler specifically produces 32-bit programs that can run under Windows 95 or Windows NT. Its 32-bit memory allocation system also greatly simplify the job of coding.

Chapter 2: The Detection Algorithm

2.1 Overview of the Algorithm

In this chapter, the details of the detection and measurement algorithms used to study spontaneous IPSCs recorded in our laboratory are described. The algorithm used is intuitive and is essentially a multi-criteria test. First of all, useful features derived from the shape of real $GABA_A$ mediated IPSCs are defined. To achieve this, 50 randomly chosen IPSCs were examined manually. The rising slope, amplitude, and the decay slope of these 50 IPSCs were measured and calculated. Then, the least values of these parameters were determined and were used as a reference for the detection algorithm.

Based on these features, three specific tests concerning the threshold, rising slope and decay slope were devised. In addition, a fast noise test was implemented. A candidate event will be considered a real event only if it passes all these criteria. Thus, although many artifacts may pass one or even two criteria, the chance that they fulfill all four criteria is small, making the algorithm a robust one. The steps involved in this process are summarised in Figure 2.1. In addition, the stringency of the criteria is user-adjustable so that the program can be used to detect IPSCs with characteristics quite different from those 50 original IPSCs. Once an event is identified, relevant parameters including the time of occurrence, rise time, peak amplitude and decay time constant are measured and reported.

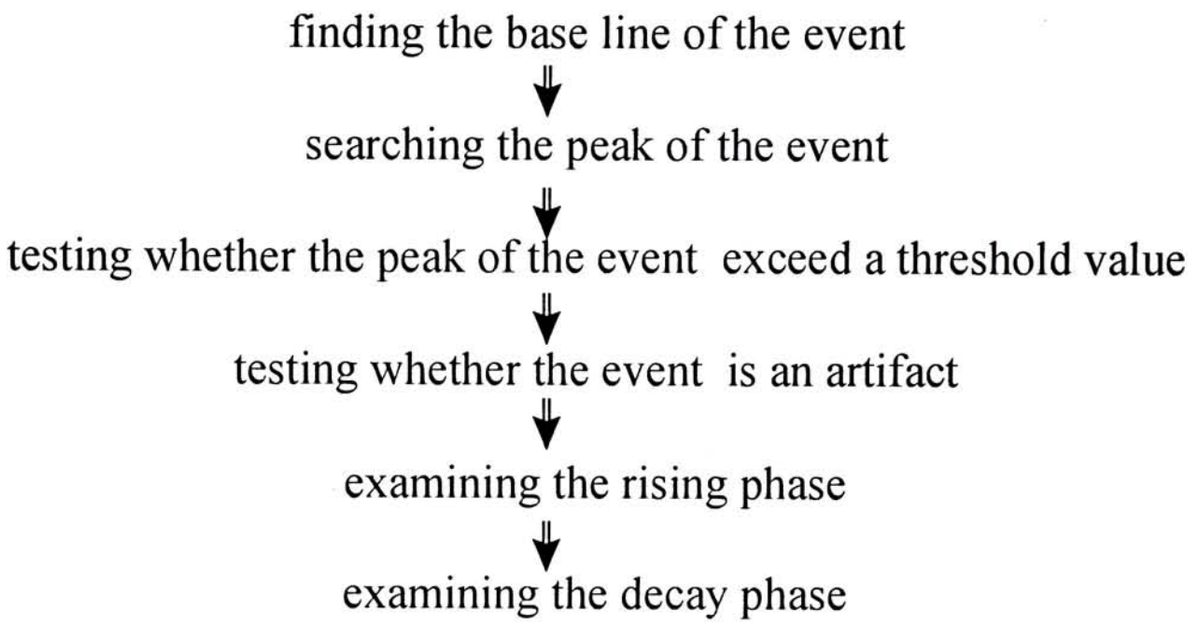


Fig. 2.1. Steps involved in detecting an event of IPSC.

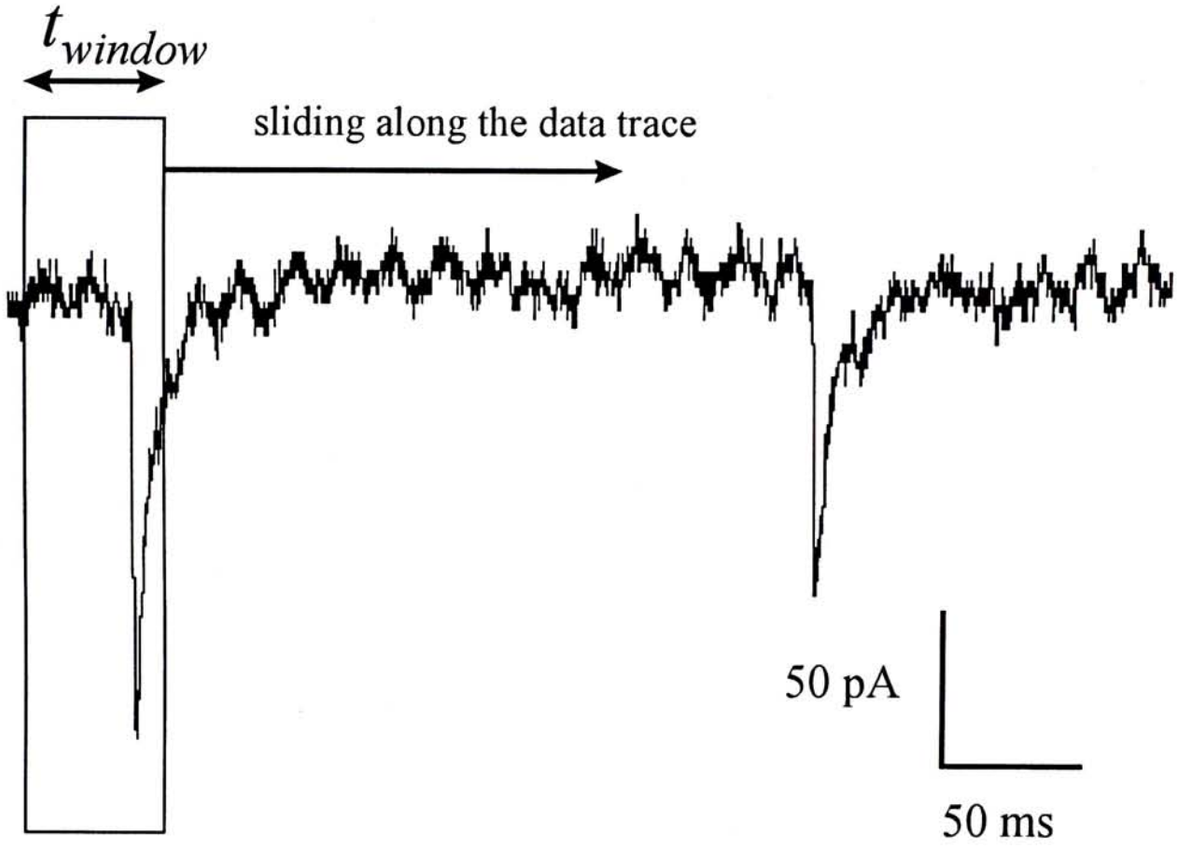
2.2 Detection of Events

The Data Window

In a single data trace as short as 2000 ms, there may already be many IPSCs. Obviously a routine that can detect a single event can be applied recurrently or repetitively to detect all the events in the whole data trace. To achieve this, a *Data Window* of duration defined by the variable t_{window} is first defined. The default value is 50 ms. The algorithm assumes that the *Data Window* contains one IPSC. Whether this assumption is true or not is tested. The *Data Window* then shifts one data point ahead (the corresponding time equal to 1/sampling rate) and the test performed again. This procedure was repeated until the end of the whole data trace. In this way, all IPSCs in the data trace will be found out (Fig. 2.2). The advantage of this approach is that it makes the codes simpler and easier to be maintained and modified although the efficiency of the overall routine may be compromised.

The Threshold Test

The presence of a candidate event is first checked by the threshold test. To find out the base line current, a portion of the data trace called the *base line portion* of the assumed IPSC is first defined. It starts at a time of $t_{base\ line\ gap}$ (default 10 ms) from the beginning of the *Data Window* and has a duration defined by the variable $t_{base\ line\ portion}$ which has a default value of 5 ms. The base line current of the IPSC, or $I_{base\ line}$, is calculated by averaging all the data points within the *base line portion* (formula 2.1).



The *Data Window*

Fig. 2.2. To detect the IPSCs in the data trace, the *Data Window* is defined. The default value of t_{window} is long enough to hold an event of IPSC. The function of the *Data Window* is to test whether there is an IPSC within it. If an IPSC exits in the *Data Window*, the IPSC is marked down. If not, nothing is done. After each testing, the *Data Window* moves one point forward and the steps repeat. When the *Data Window* arrives at the end of the data trace, all the IPSCs are marked.

$$I_{base\ line} = \frac{\sum_{i=1}^{n} I_i}{n} \qquad (2.1)$$

where n is the number of data points falling into the *base line portion*, and I_1 to I_n are the values of the currents of the individual data points.

In a similar manner, a portion of the data trace that starts at a time of $t_{peak\ gap}$ from the beginning of the *Data Window* is defined to be the *peak portion* of the IPSC. The default value of $t_{peak\ gap}$ is equal to 20 ms. The duration of the *peak portion* is defined by the variable $t_{peak\ portion}$. Its default value is 5 ms (Fig. 2.3). Each data point within this *peak portion* is averaged with its previous and the next points to eliminate the background noise according to formula 2.2.

$$I'_i = \frac{I_{i-1} + I_i + I_{i+1}}{3} \qquad (2.2)$$

where I_i is the value of the current of data point i and I'_i is the averaged value of current of data point i. Here, 3 data points are taken for averaging is a compromise between the effectiveness of eliminating the noise and the degree of distortion of the data caused by averaging. The peak current of the assumed IPSC, $I_{cand\ peak}$, is then defined as the most negative averaged current among these data points. If there are more than one such data point, the one nearest to the beginning of the *peak portion* is selected. The peak found is designated as $P_{cand\ peak}$. If $I_{base\ line} - I_{cand\ peak}$ is greater than or equal to the threshold amplitude, $I_{threshold}$, the IPSC is considered to be a *candidate event* (Fig. 2.4). The value of $I_{threshold}$ is set and can be changed by the user according to the noise level of the

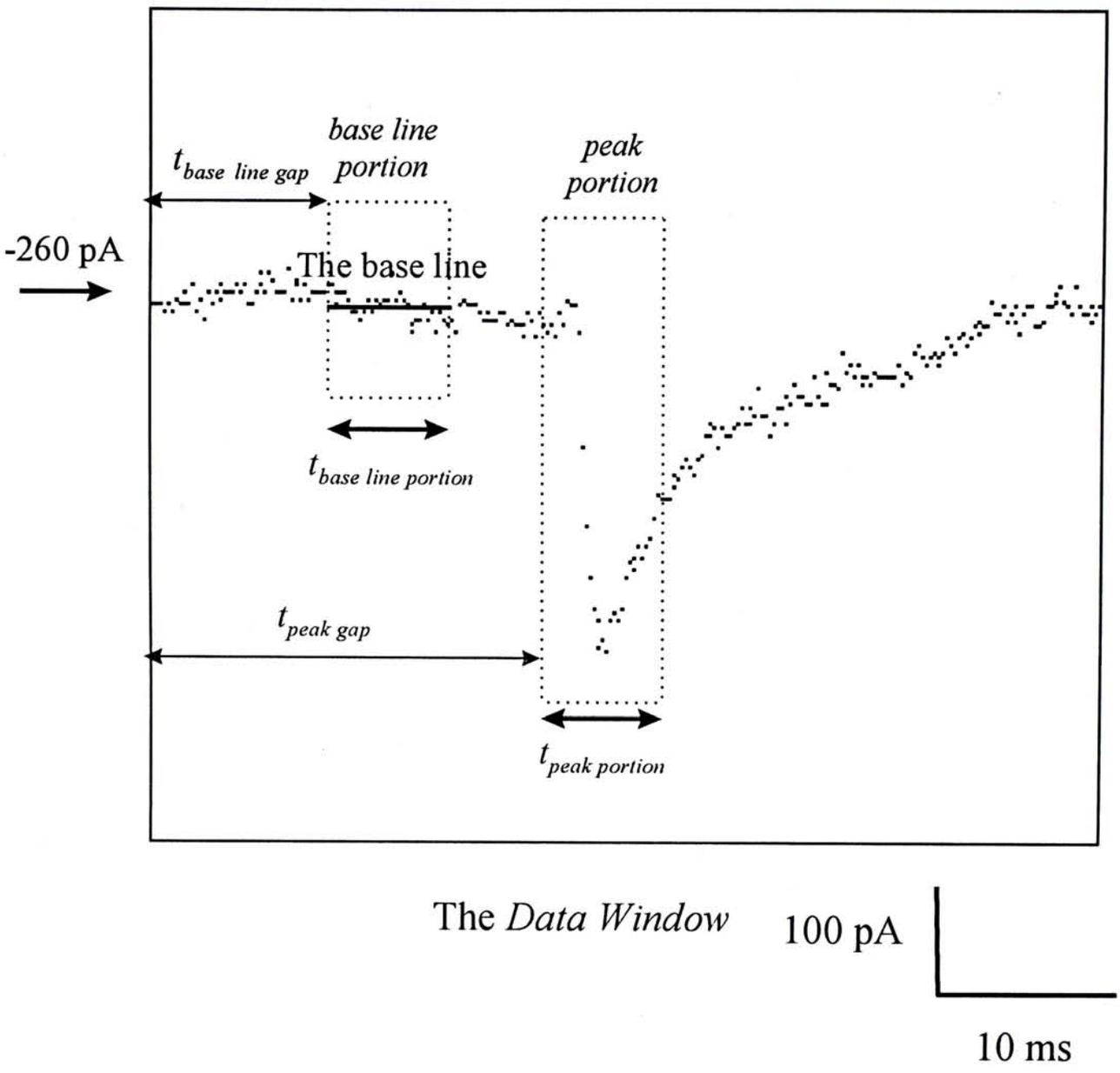


Fig. 2.3. Within the *Data Window*, the *base line portion* and the *peak portion* are defined to find out the base line current and the amplitude of the assumed IPSC respectively. The length of the *base line portion* is long enough to find out the averaged base line currents. Its relative position within the *Data Window* can be changed by adjusting the variable $t_{base\ line\ gap}$. The length of the *peak portion* is long enough to include the whole peak region. Its relative position within the *Data Window* can be changed by adjusting the variable $t_{peak\ gap}$.

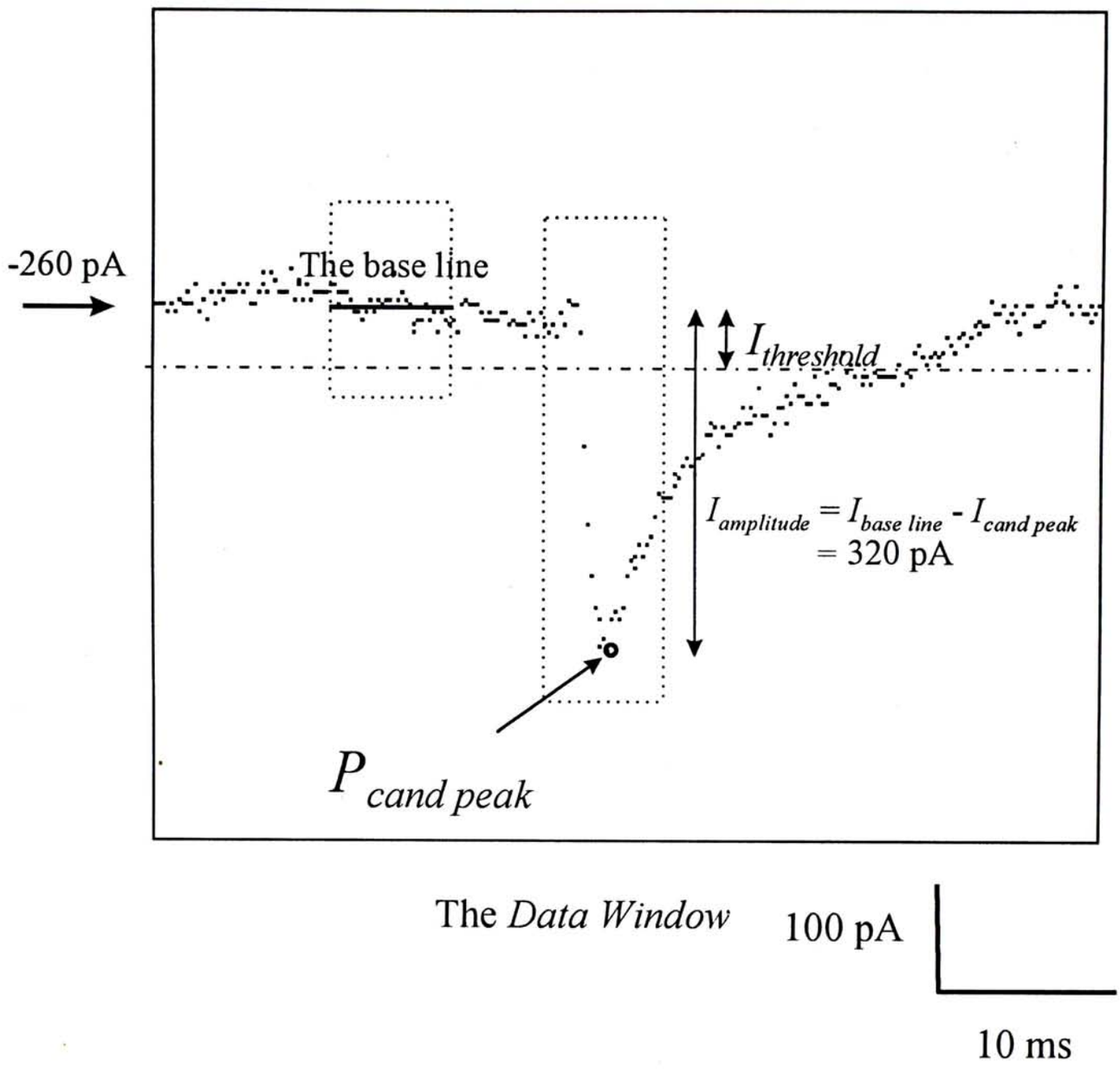


Fig. 2.4. To find out the base line current, $I_{base\ line}$, all the data points within the *base line portion* are averaged and then assigned to be the base line current. The peak current of the IPSC, $I_{cand\ peak}$, is equal to the most negative averaged current among those within the *peak portion*. In this example, the value of the threshold amplitude, $I_{threshold}$ is set to be 50 pA. As $I_{base\ line} - I_{cand\ peak}$ exceeds $I_{threshold}$, the most negative averaged data point is assigned to be the peak of a *candidate event*, $P_{cand\ peak}$. The amplitude of the *candidate event*, $I_{amplitude}$, is assigned to be 320 pA.

data. The amplitude of the *candidate event* is designated by $I_{amplitude}$ which is equal to $I_{base\ line} - I_{cand\ peak}$.

The Fast Noise Test

As mentioned before, a candidate event will be considered a real synaptic event only if it passes additional criteria. The first of these is the fast noise test. This is because fast noises are quite common in electrophysiological recordings. The criterion is based on the fact that a fast noise is extremely short in duration. If the amplitude of the signal drops back to baseline quickly, it is very likely that it is a fast noise.

To implement the test, a portion of the data trace immediately following the $P_{cand\ peak}$ with length equal to $t_{peak\ width}$ is first defined. The default value of $t_{peak\ width}$ is 1 ms. This portion of data trace is designated as *peak width*. Then, $I_{displacement}$, the difference between the current value of the data points within *peak width* and the $I_{base\ line}$, is obtained according to formula 2.3.

$$I_{displacement\ i} = I_{base\ line} - I_i \qquad (2.3)$$

where I_i is the current of data point i within *peak width* and $I_{base\ line}$ is the base line current. The *candidate event* is considered not be a fast noise if and only if within *peak width*, there are 80% or more data points which $I_{displacement}$ is greater than or equal to 0.5 $I_{amplitude}$ of the *candidate event* (Fig. 2.5). The 80% and the 0.5 $I_{amplitude}$ setting is empirical, based on observations that human defined events, but not fast noises, always pass this criterion.

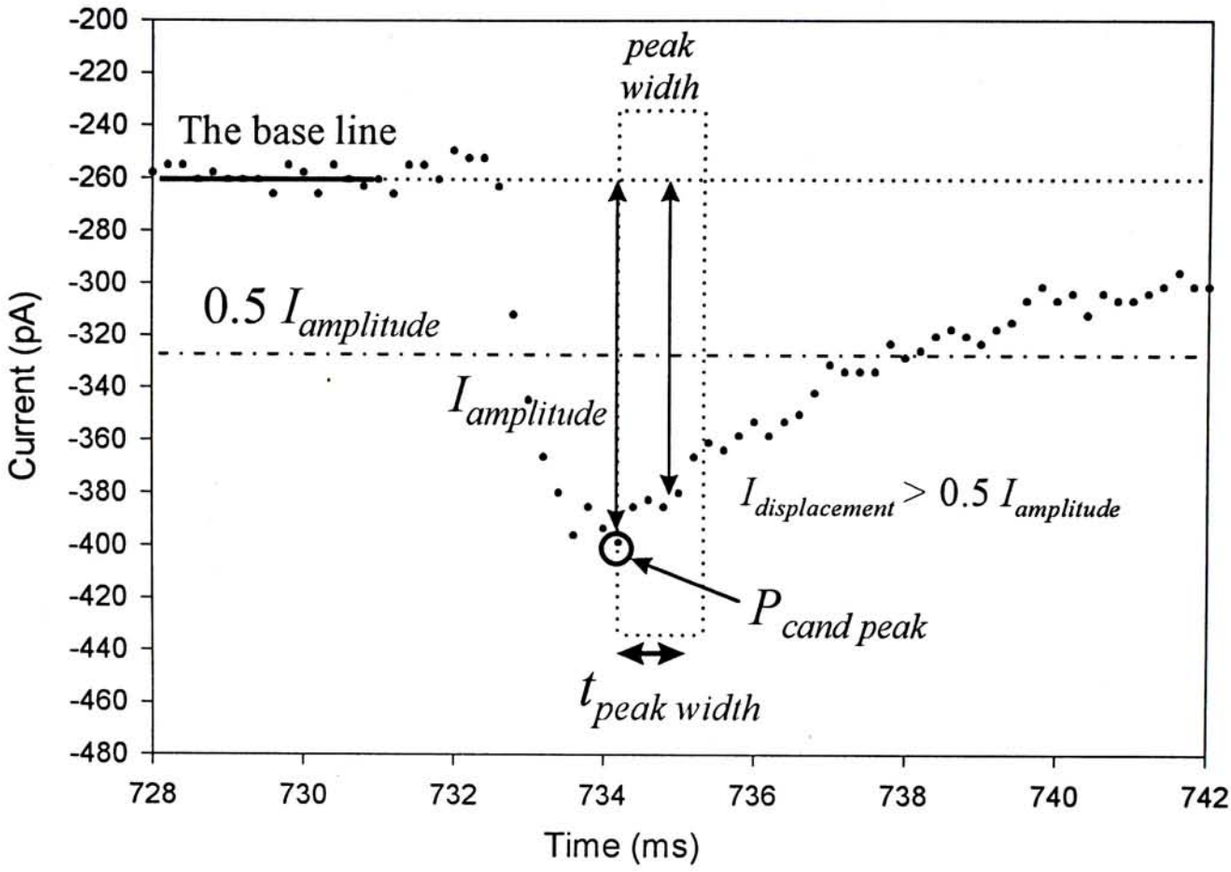


Fig. 2.5a. A true IPSC passes the fast noise test. The default value of $t_{peak\ width}$ is 1 ms. In the above figure, there are five data points in the *peak width* defined immediately after the $P_{cand\ peak}$. The $I_{displacement}$ of all these five data points are greater than 0.5 $I_{amplitude}$. The percentage of data points which passes this criterion is therefore 100%. As a result, this IPSC is considered to have passed the fast noise test.

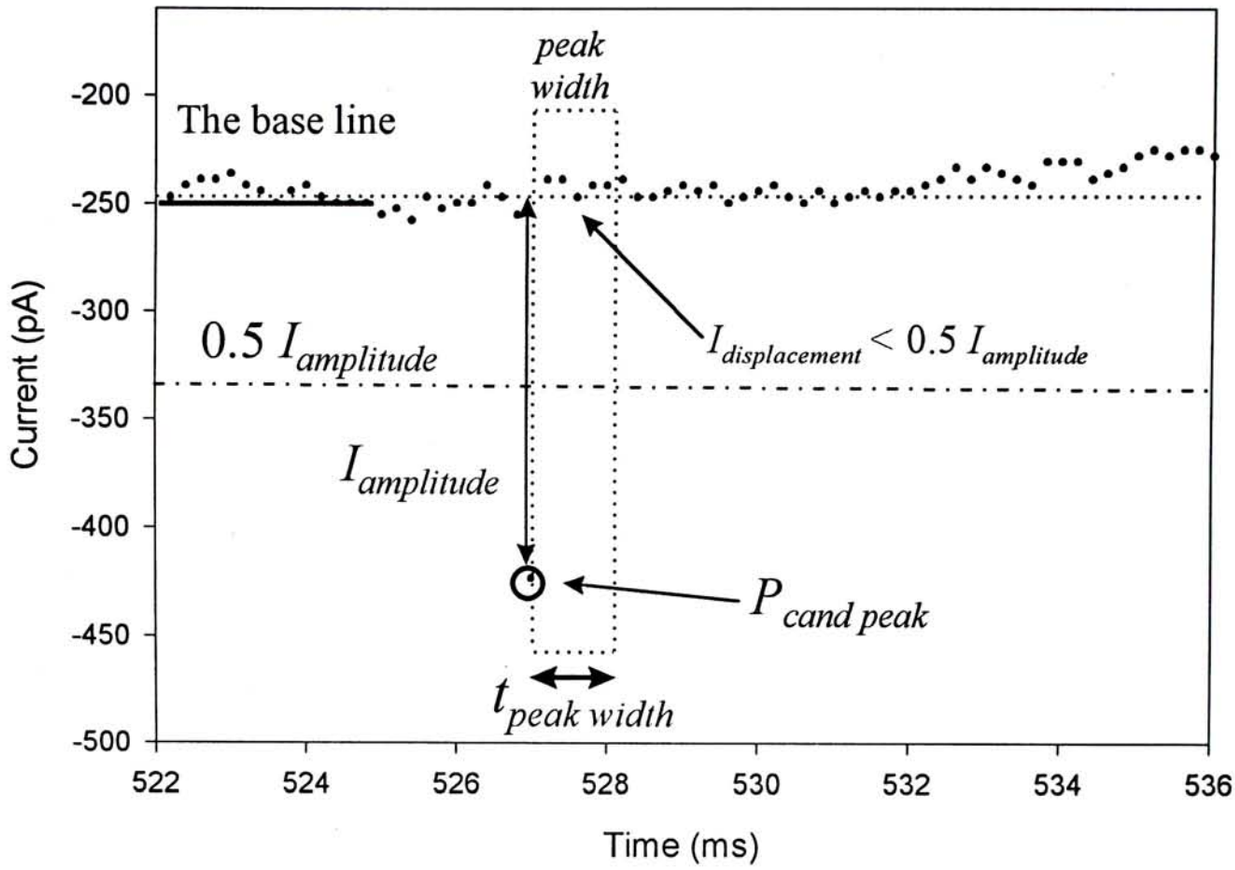


Fig. 2.5b. A fast noise is eliminated by the test. The default value of $t_{peak\ width}$ is 1 ms. In the above figure, there are five data points in the *peak width*. The $I_{displacement}$ of all these five data points are smaller than 0.5 $I_{amplitude}$. The percentage of data points which passes this criterion is therefore 0%. As a result, the noise is eliminated by the test.

Testing the Rising Slope

To test whether the rising slope of a *candidate event* exceeds a threshold slope, a portion of data trace immediately before the $P_{cand\ peak}$ is defined to be the *rising portion* of the *candidate event*. The length of the *rising portion* is defined by the variable $t_{rising\ portion}$. Its default value is 4 ms. This length is long enough to hold the rising phase of very slow IPSCs.

The slope of each data points within the *rising portion* is then calculated by formula 2.4

$$S_i = \frac{I_{i+1} - I_i}{\Delta t} \qquad (2.4)$$

where S_i is the slope of data point i and I_i is the current of data point i within the *rising portion*. Δt is the time interval between two successive data points.

As a *true event* usually has a rapid rising phase, a threshold rising slope, $S_{rising\ threshold}$, is defined to distinguish a *true event* and a false event. The value of $S_{rising\ threshold}$ is set by the users during detection and is subjected to change according to the kinetic characteristic of the interested IPSCs. All the data points within the *rising portion* are tested whether their slopes are more negative than $S_{rising\ threshold}$. Then, the number of data points the slopes of which satisfy this condition is found out.

True IPSCs usually have more than one data point the slope of which satisfy this condition when the sampling rate is higher than 2.5 kHz. However, fast noise may only has one data point satisfying this condition. To distinguish the true IPSCs and the fast noises, the threshold number of data points the

rising slopes of which are more negative than the threshold rising slope is defined. It is designated as $N_{rising\ threshold}$. If out of the total number of data points within the *rising portion*, the number of data points the slopes of which pass the criteria is equal to or greater than $N_{rising\ threshold}$, the *candidate event* is considered to have passed the rising phase criteria.

The default value of $N_{rising\ threshold}$ is 2. If there are 2 or more data points whose slope are more negative than the threshold rising slope, the *candidate event* passes the rising phase criteria. However, to make the Automatic Detection software more flexible to detect IPSCs with different rising time, $N_{rising\ threshold}$ can be changed by the users according to their needs (Fig. 2.6).

Testing the Decay Slope

To test whether the decay slope of a *candidate event* exceeds a threshold slope, a portion of data trace immediately following the $P_{cand\ peak}$ is defined to be the *decay portion* of the *candidate event*. The length of the *decay portion* is defined by the variable $t_{decay\ portion}$. Its default value is 3 ms. This length of the *decay portion* only considers the first part of the decay phase instead of the whole decay phase. The advantage is that if the later part of the decay phase is overlapped by another event, no significant error will occur.

The slope of each data point within the *decay portion* is examined. As the background noises may cause fluctuation of the current trace and affect the accurate estimation of the slope, the data points are first averaged with two points before and two points after the data points before calculating the slope (formula 2.5).

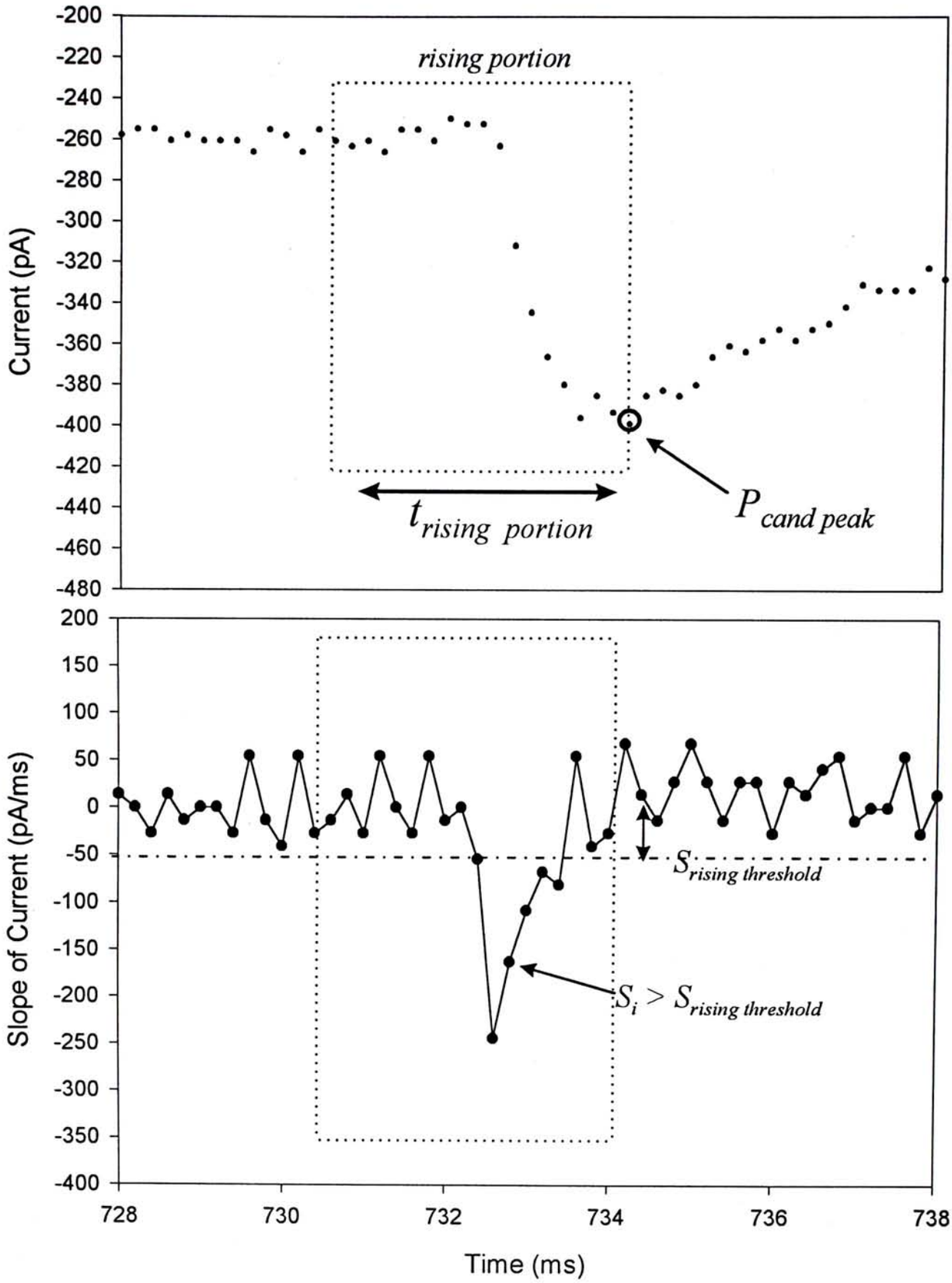


Fig. 2.6a. A true IPSC passes the rising phase criterion. In the above figure, the threshold rising slope $S_{rising\ threshold}$ is set to be -50 nA/s. The default value of $t_{rising\ portion}$ is 4 ms. There are 17 data points in the *rising portion*. Out of the 17 data points, there are 6 data points the slopes of which, S_i, exceed $S_{rising\ threshold}$. The default value of $N_{rising\ threshold}$ is 2. Therefore, the number of data points the slopes of which exceed the threshold rising slope is greater than $N_{rising\ threshold}$. As a result, this IPSC is considered to have passed the rising phase criterion.

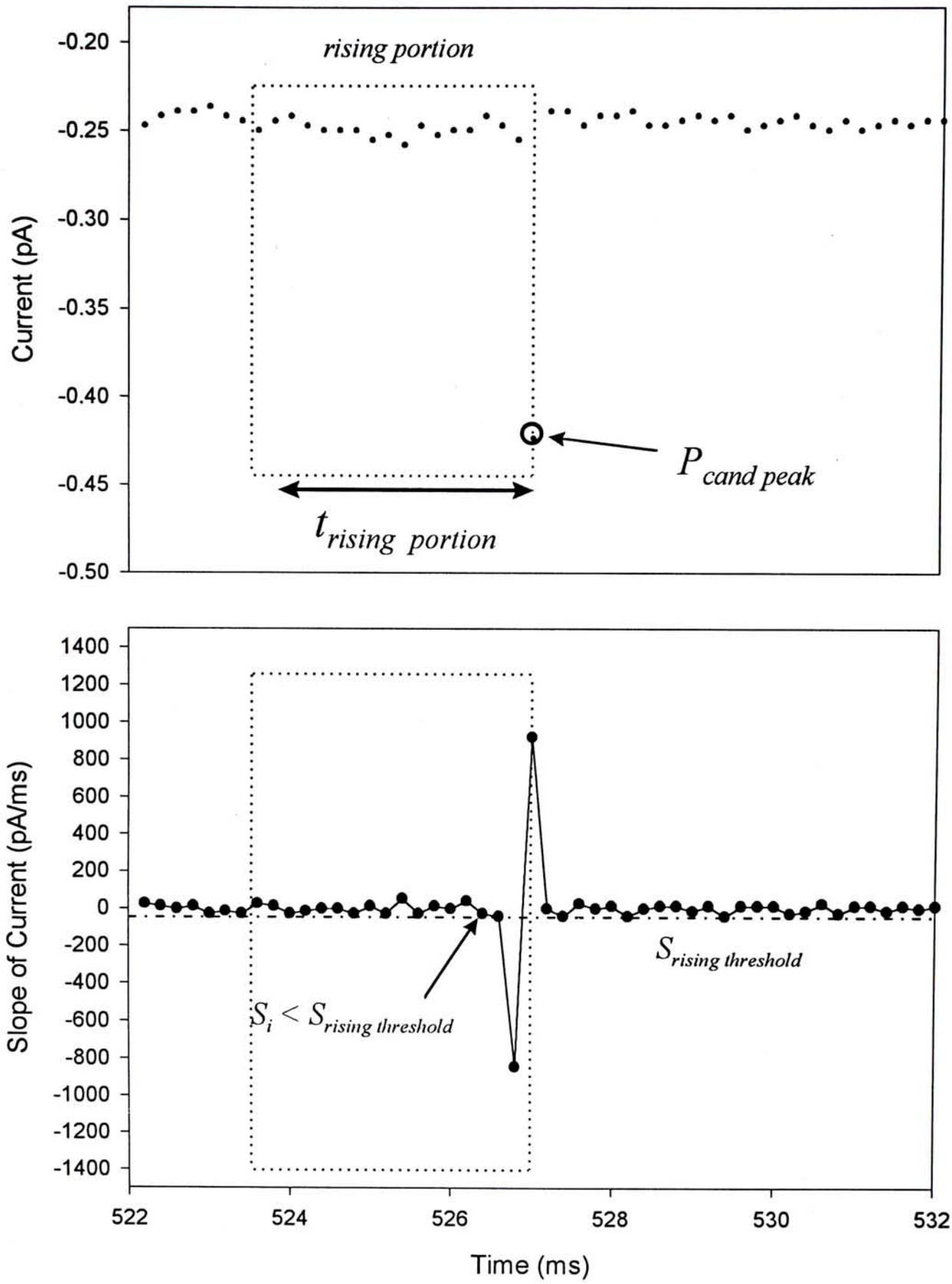


Fig. 2.6b. A false IPSC is eliminated by the rising phase criterion. In the above figure, the threshold rising slope $S_{rising\ threshold}$ is set to be -50 nA/s. The default value of $t_{rising\ portion}$ is 4 ms. There are 17 data points in the *rising portion*. Out of the 17 data points, there is only 1 data point the slope of which, S_i, exceed $S_{rising\ threshold}$. The default value of $N_{rising\ threshold}$ is 2. Therefore, the number of data points the slope of which exceed the threshold rising slope is smaller than $N_{rising\ threshold}$. As a result, this false IPSC is eliminated by the rising phase criterion.

$$I''_i = \frac{I_{i-2} + I_{i-1} + I_i + I_{i+1} + I_{i+2}}{5} \qquad (2.5)$$

where I''_i is the averaged current of the data point i, and I_i is the current of data point i.

The slope of the data point, S'', is then calculated by formula 2.6.

$$S''_i = \frac{I''_{i+3} - I''_{i-3}}{6\Delta t} \qquad (2.6)$$

where S''_i is the slope of data point i, I''_{i+3} and I''_{i-3} are the averaged current of 3 points after and 3 points before data point i respectively. Δt is the time interval between 2 successive data points. This method gives a better estimation of the decay slope even when the data trace is very noisy as the data points are averaged before their slopes are calculated.

To confirm a *candidate event* being a *true event*, a threshold decay slope, $S''_{decay\ threshold}$, is defined. The value of $S''_{decay\ threshold}$ is defined by the user during execution. All the data points within the *decay portion* are tested whether their slopes exceed $S''_{decay\ threshold}$. Theoretically, the slopes of all the data points within the *decay portion* should satisfy this condition in order to be a true IPSC. However, practically, noises may cause fluctuation of the slope of the data points. As a result, some data points' slopes of the true IPSCs may not exceed $S''_{decay\ threshold}$. Therefore, in practice, not all data points' slopes are required to satisfy this condition. In stead, the ratio between the number of data points whose S'' exceed the threshold decay slope and the total number of data points in the *decay portion* is found out. A threshold ratio is then defined to distinguish true IPSCs and false IPSCs. The threshold ratio is designated as $R_{decay\ threshold}$.

If the ratio between the number of data points the S'' of which exceed the threshold decay slope and the total number of data points in the *decay portion* is equal to or larger than $R_{decay\ threshold}$, the *candidate event* is considered to have passed the decay phase criteria.

The default value of $R_{decay\ threshold}$ is 0.5. Therefore, if there are 50% of data points whose slope exceed $S''_{decay\ threshold}$, the *candidate event* passes the decay phase test. The default value of $R_{decay\ threshold}$ is empirically defined. All observed IPSCs pass this default value. The value of $R_{decay\ threshold}$ are also subjected to be changed according to the needs of the users. This make the Automatic Detection software more flexible to detect IPSCs with different decay slope. (Fig. 2.7).

Test of Events Being Overlapping

After all of the true events in the data trace are detected, whether these events overlap with each other is tested. To achieve this, the *event range* and the *base line range* of each event are defined. The *event range* of an IPSC is defined to be the region of the data trace between 10% rising and 10% decay of the IPSC. To find out the 10% rising and 10% decay of the event, the currents of the data points are averaged to eliminate noise fluctuation. Then, $I'_{displacement}$, the difference between the averaged current of the data points and the $I_{base\ line}$, is calculated (formula 2.7).

$$I'_{displacement\ i} = I_{base\ line} - I'_i \qquad (2.7)$$

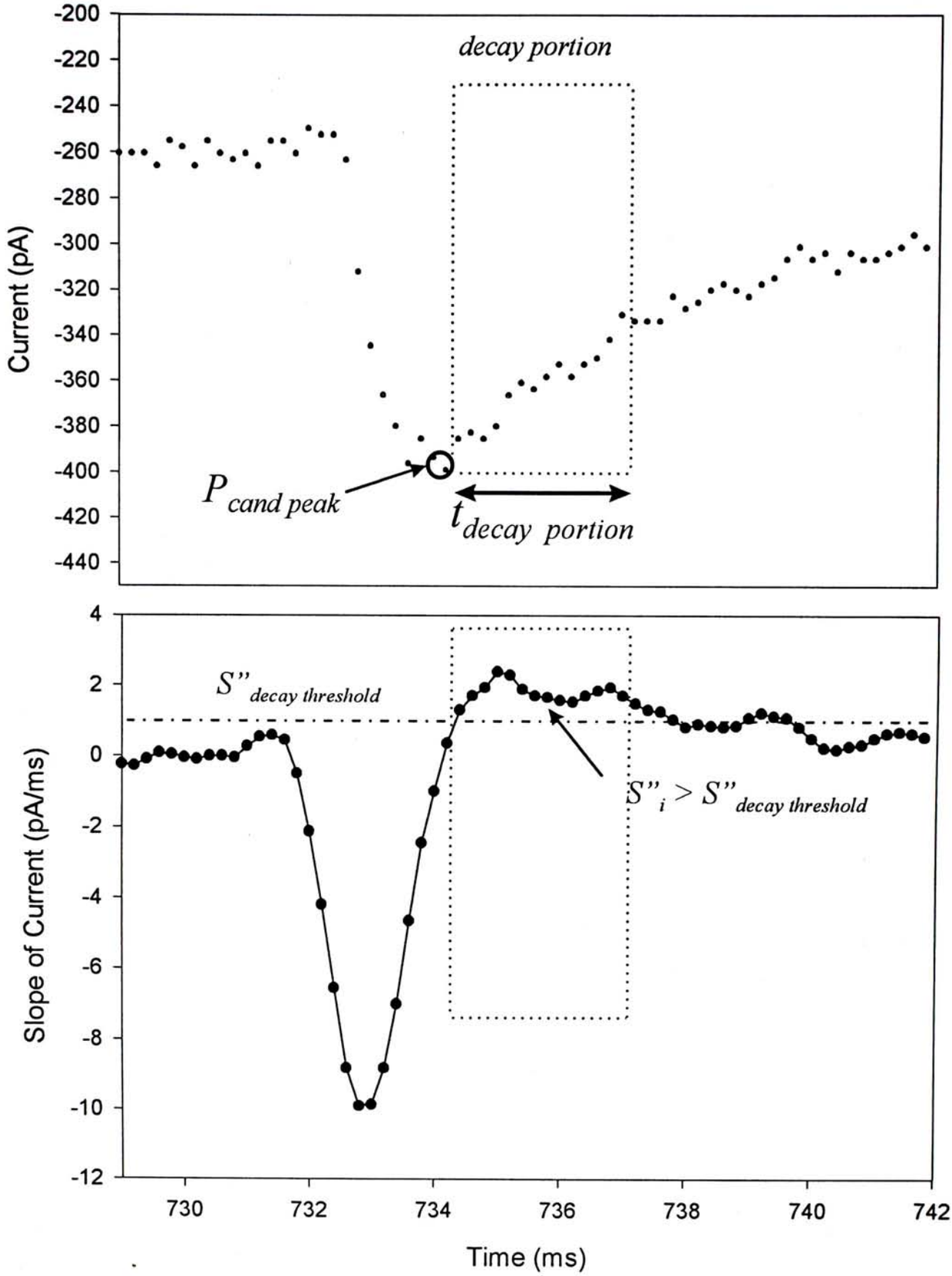


Fig. 2.7a. A true IPSC passes the decay phase test. In the above figure, the threshold decay slope $S''_{decay\ threshold}$ is set to be 1 pA/ms. The default value of $t_{decay\ portion}$ is 3 ms. There are 14 data points in the *decay portion*. Out of these 14 data points, all these 14 data points' slope S'' exceed $S''_{decay\ threshold}$. The ratio between the number of data points the S'' of which exceed the threshold decay slope and the total number of data points in the *decay portion* is therefore equal to 1. As the default value of $R_{decay\ threshold}$ is 0.5, this ratio is greater than $R_{decay\ threshold}$. As a result, this IPSC is considered to have passed the decay phase test.

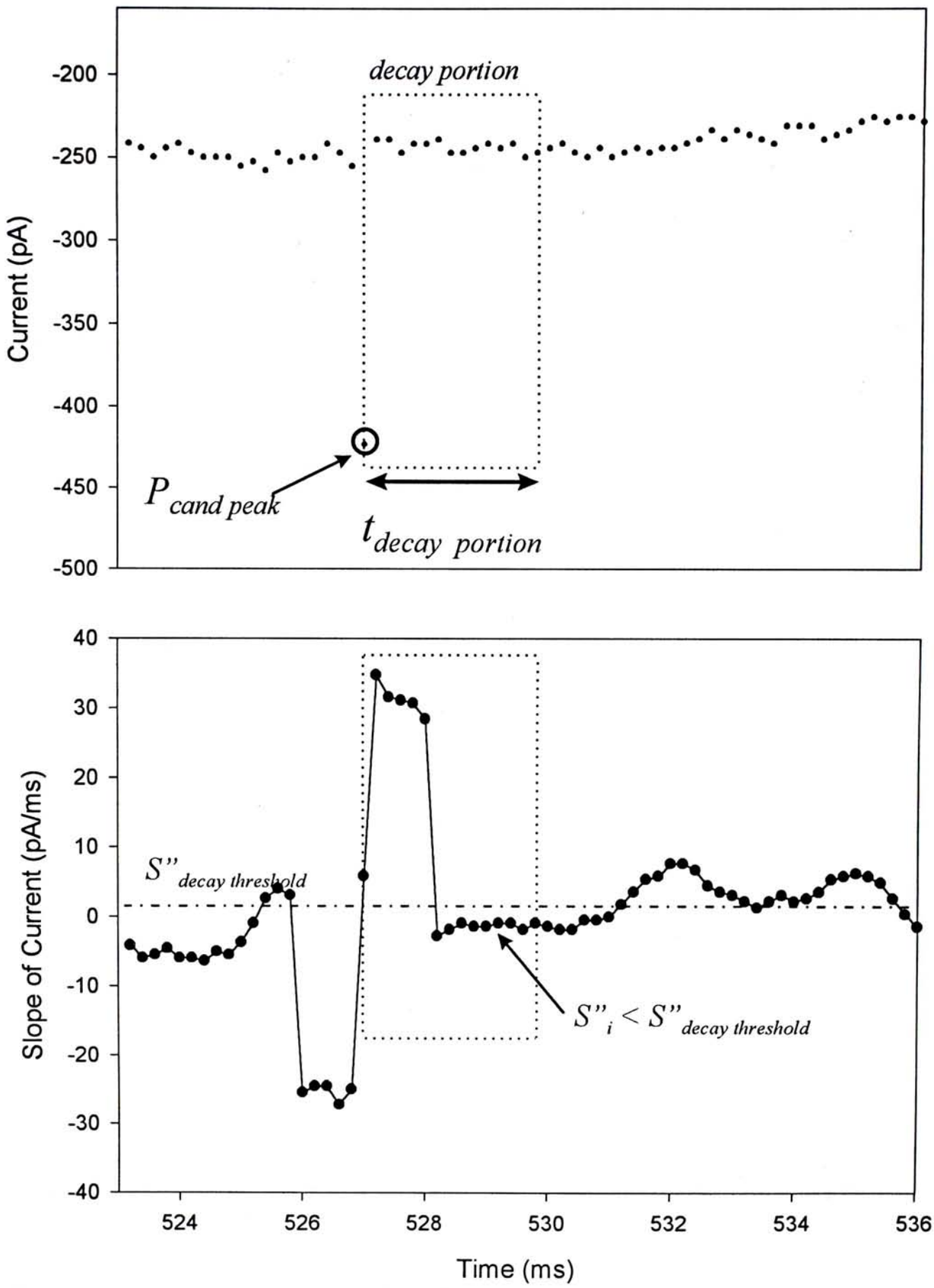


Fig. 2.7b. A fast noise is detected by the decay phase test. In the above figure, the threshold decay slope $S''_{decay\ threshold}$ is set to be 1 pA/ms. The default value of $t_{decay\ portion}$ is 3 ms. There are 14 data points in the *decay portion*. Out of these 14 data points, 5 data points' slope S'' exceed $S''_{decay\ threshold}$. The ratio between the number of data points the S'' of which exceeds the threshold decay slope and the total number of data points in the *decay portion* is therefore equal to 0.36. As the default value of $R_{decay\ threshold}$ is 0.5, this ratio is smaller than $R_{decay\ threshold}$. As a result, this fast noise is rejected by the decay phase test.

where I'_i is the averaged current of data point i (for the calculation of I'_i, see formula 2.2) and $I_{base\ line}$ is the base line current of the event.

The backward limit of the event, or $E_{back\ limit}$, is defined to be the data point which corresponds to 10% rise time of the IPSC. To locate the $E_{back\ limit}$, the data trace is scanned from the peak toward left until a data point whose $I'_{displacement}$ is equal to 10% of $I_{amplitude}$ of the IPSC is met. This data point is then defined to be the $E_{back\ limit}$ of the IPSC. The forward limit of the event, or $E_{for\ limit}$, is found in a similar way. The portion of data trace between $E_{back\ limit}$ and $E_{for\ limit}$ is then defined as the *event range* of the IPSC (Fig. 2.8).

The *base line range* of an IPSC is defined to be a portion of data trace immediately before the peak of the IPSC. The length of the *base line range* is defined by the variable $t_{base\ line\ range}$, whose default value is 20 ms (Fig. 2.9). The value of $t_{base\ line\ range}$ is subjected to be changed when it is required. This make the Automatic Detection software more flexible to define overlapping events. The default value $t_{base\ line\ range}$ is arbitrarily defined. This value is long enough so that no overlapping events will be missed.

After defining the *event range* and the *base line range* of each IPSCs, whether successive IPSCs overlap with each other is tested. Let $P_{peak\ x}$ and $P_{peak\ y}$ denote two successive events of IPSCs. $P_{peak\ y}$ is considered to have overlapped with $P_{peak\ x}$ if there are one or more data points which fall into both the *event range* of $P_{peak\ x}$ and the *base line range* of $P_{peak\ y}$. (Fig. 2.10).

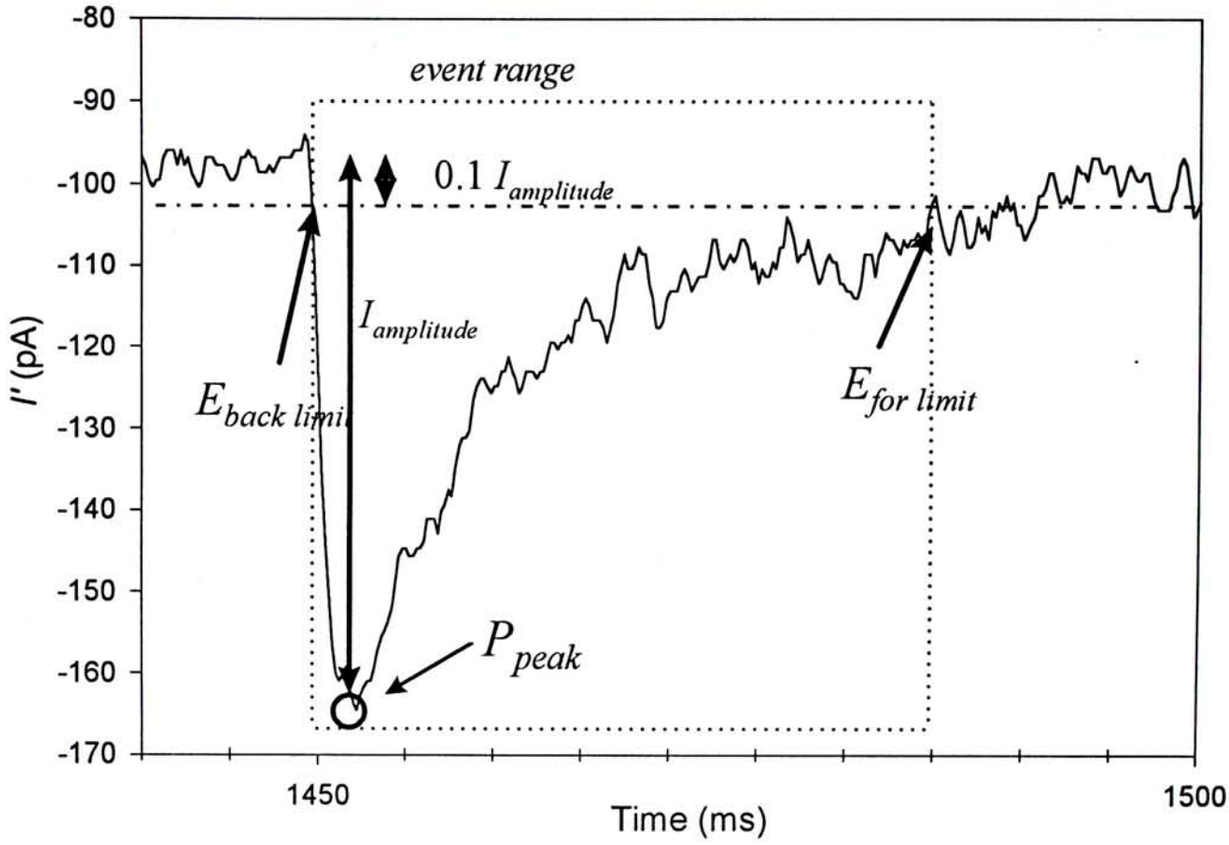


Fig. 2.8. To define the *event range*, the backward limit, $E_{back\ limit}$, and the forward limit, $E_{for\ limit}$, of the IPSC are defined. The $E_{back\ limit}$ cooresponds to the 10% rise time and the $E_{for\ limit}$ cooresponds to the 10% decay time. The portion of data trace between these *two limits* is the *event range* of the IPSC.

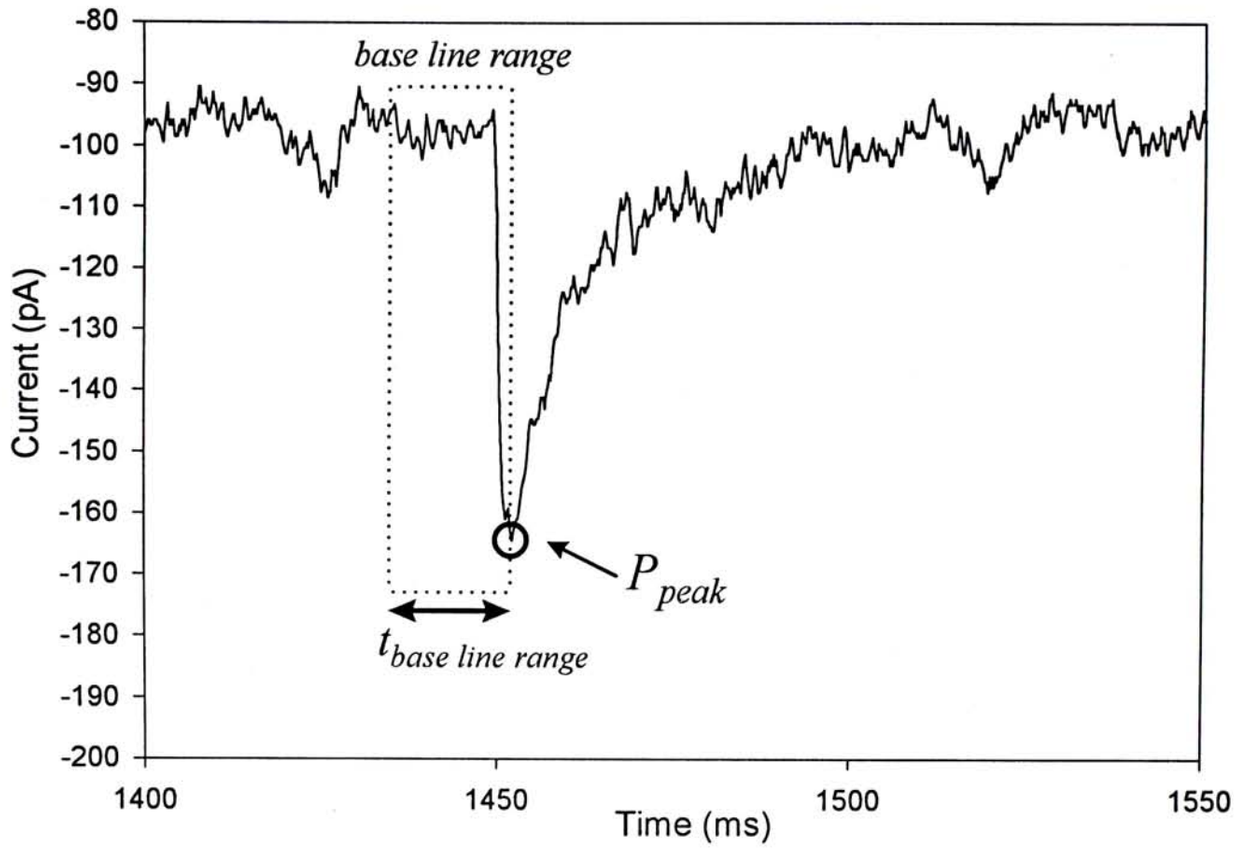


Fig. 2.9. To define the *base line range*, a portion of data trace immediately before P_{peak} is marked as the *base line range*. Its default length is 20 ms.

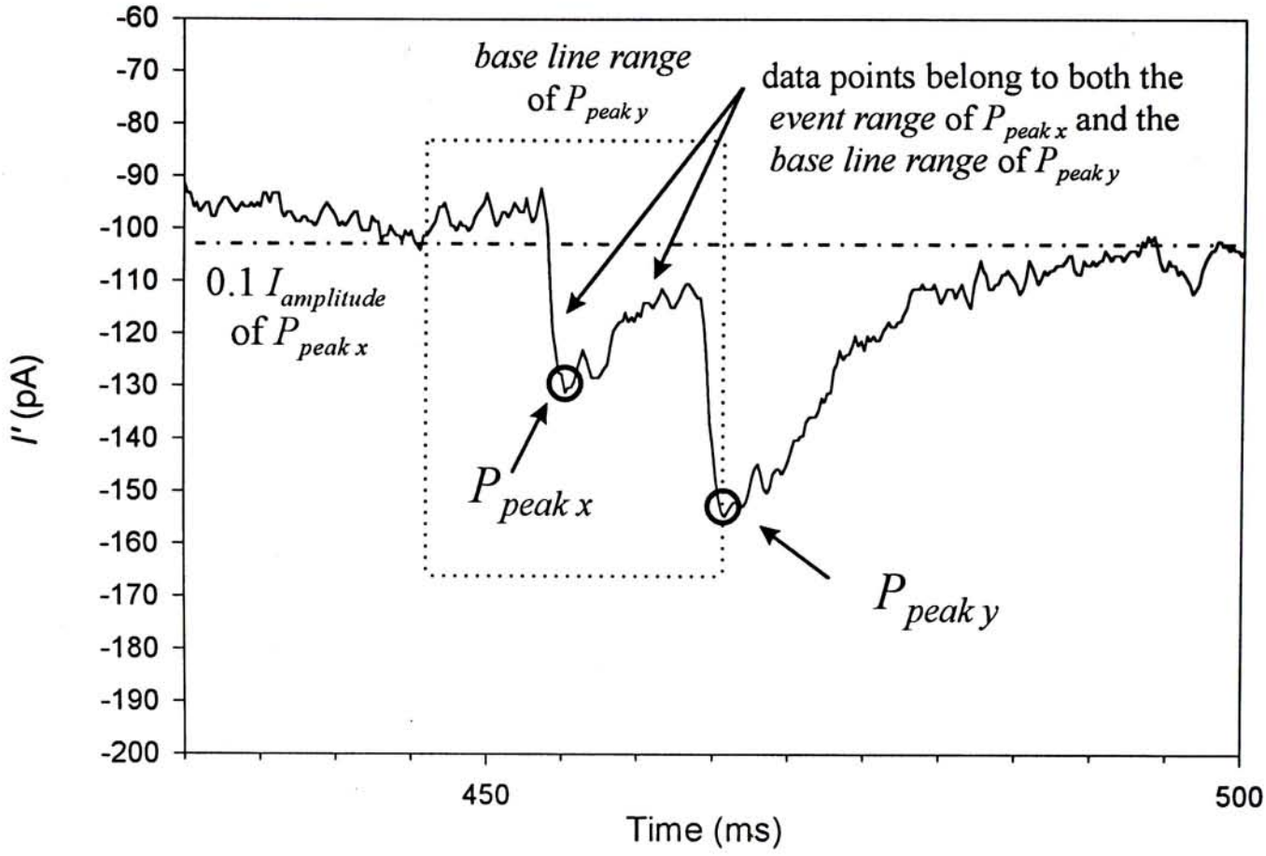


Fig. 2.10. In the above diagram, the *base line range* of $P_{peak\,y}$ is marked by the dotted rectangle. Within the *base line range* of $P_{peak\,y}$, there are data points which also belong to the *event range* of $P_{peak\,x}$. Therefore, $P_{peak\,y}$ is considered to have overlapped with $P_{peak\,x}$.

After all the overlapping events are found out, the percentage of overlapping event is calculated by dividing the number of overlapping events with the total event number.

2.3 Measurement of Parameters

After the events were detected, the next step was measurement of parameters of the IPSCs. When successive events overlap together, the events may interfere the measurement of the parameters of other events. Therefore, only the parameters of non-overlapping events are measured. The measured parameters included the rise times and decay time constants.

Measuring the Rise Time

To characterize the rising phase of an IPSC, its 20% to 80% rise time are measured. To locate the 80% rise time, the data trace is scanned from the peak toward left until a data point whose $I_{displacement}$ is equal to or less than 80% of $I_{amplitude}$ of the IPSC is met. This data point is then defined to be the 80% rise time of the IPSC. The 20% rise time is found similarly, the data trace is scanned from the peak toward left until a data point which $I_{displacement}$ is equal to or less than 20% of $I_{amplitude}$ of the IPSC is met. This data point is then defined to be the 20% rise time of the IPSC. The 20%-80% rise time of the IPSC is then defined to be the time difference between these two data points (Fig. 2.11).

Measuring the Decay Time Constant

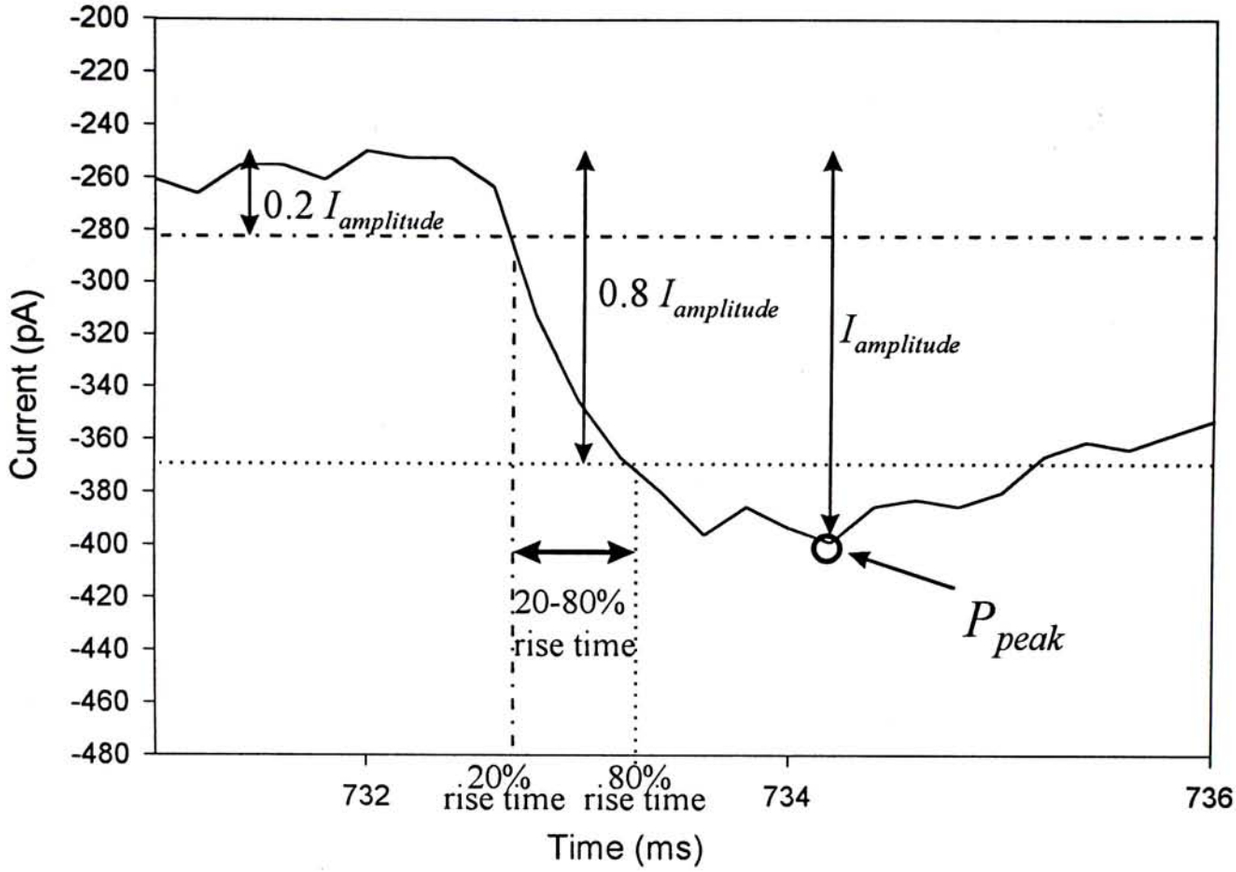


Fig. 2.11. The 20%-80% rise time of the IPSC is the time difference between the 20% rise time and the 80% rise time.

To characterize the decay time constant of an IPSC, its decay phase are fitted with a single exponential function using the least-square fitting method. To do this, the 90% decay time and 10% decay time are first found. The procedures involved is similar to the one used in finding out the 20% and 80% rise time. Then, the portion of data trace within these two data points are fitted with the following single exponential function (formula 2.8).

$$I = \mathrm{A} + \mathrm{B}e^{\frac{-t}{\tau}} \qquad (2.8)$$

where I is the ideal decay current, A is the base line current, B is the amplitude, τ is the decay time constant and t is the time (Fig. 2.12).

The τ is substituted from τ_{min} to τ_{max} to find out the mean error square between the exponential function and the value of the data points in data trace. The default value of τ_{min} and τ_{max} are set to be 0 ms and 50 ms respectively as the decay time constants of the IPSCs are not likely to exceed this range. The τ which generates the least mean error square is then taken as the decay time constant of the IPSC.

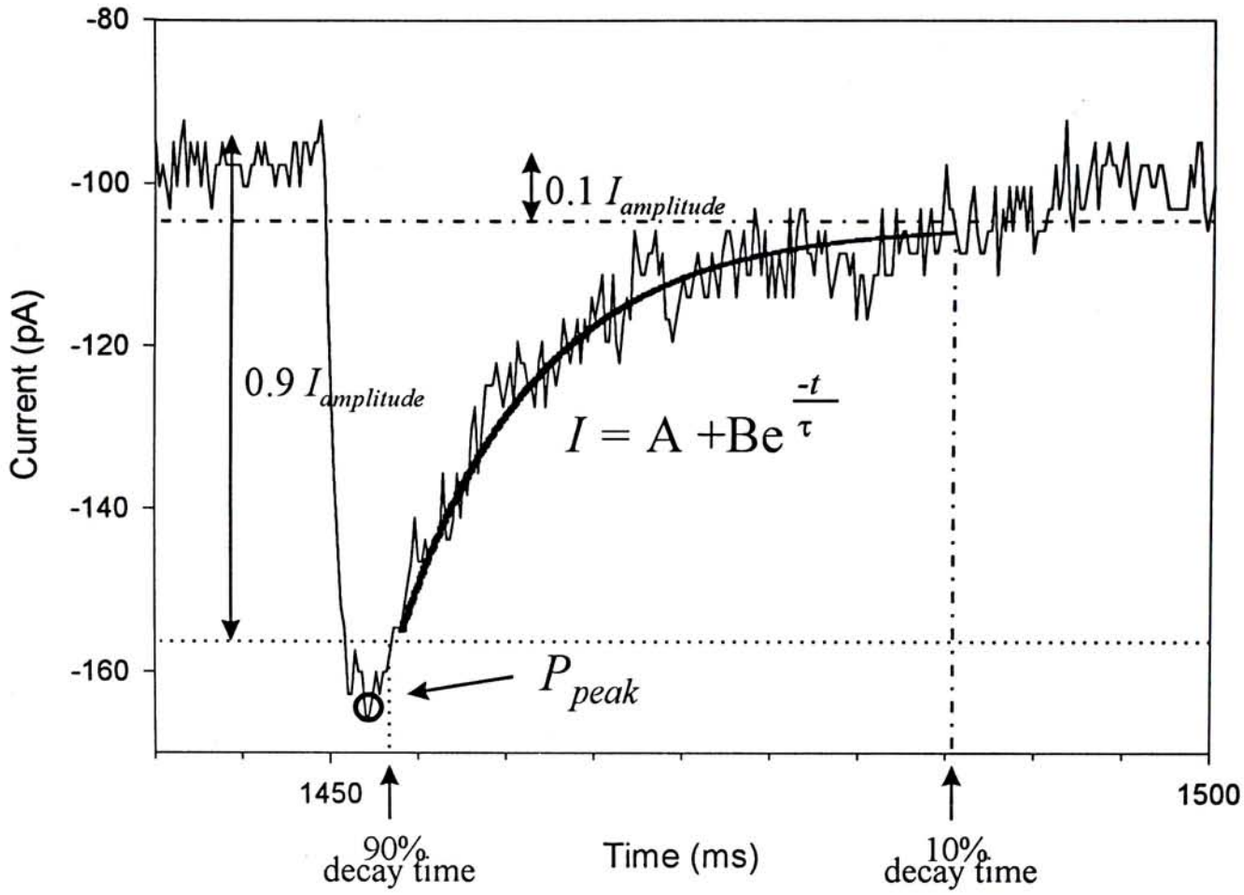


Fig. 2.12. To find out the decay time constant, the portion of data trace within the 90% decay time and 10% decay time is chosen to be fitted with the single exponential function. This portion of data trace covers most of the decay phase of the IPSC.

Chapter 3: Performance Evaluation

This chapter concentrates on the evaluation of the Automatic Detection program developed for the detection of spontaneous IPSCs. The basic features of the program are first described. This is followed by a discussion of the method of evaluating the performance. The results of evaluation are then reported and discussed.

3.1 The Automatic Detection Software

The Automatic Detection program is a 32-bit Windows based software (Fig. 3.1). It was run under Windows 95 on a Pentium 166 computer to detect $GABA_A$ receptor mediated IPSCs recorded in our laboratory. Users can use the software to view the data collected by the CED Patch and Voltage Clamp software, and select the interested sections of data for detection of IPSCs. The detected IPSCs are marked on screen with circles. (Fig. 3.2). It displays graphically the measured base line current, 20% rise time, 80% rise time and the half decay time of the non-overlapping events. It also displays the results of fitting the decay phase with the single exponential function (Fig. 3.3). After finding out the total number of events, the program calculates the frequency of the events. All these parameters can be saved as text files and be imported into other software for statistical analysis and generation of graphs (Fig. 3.4).

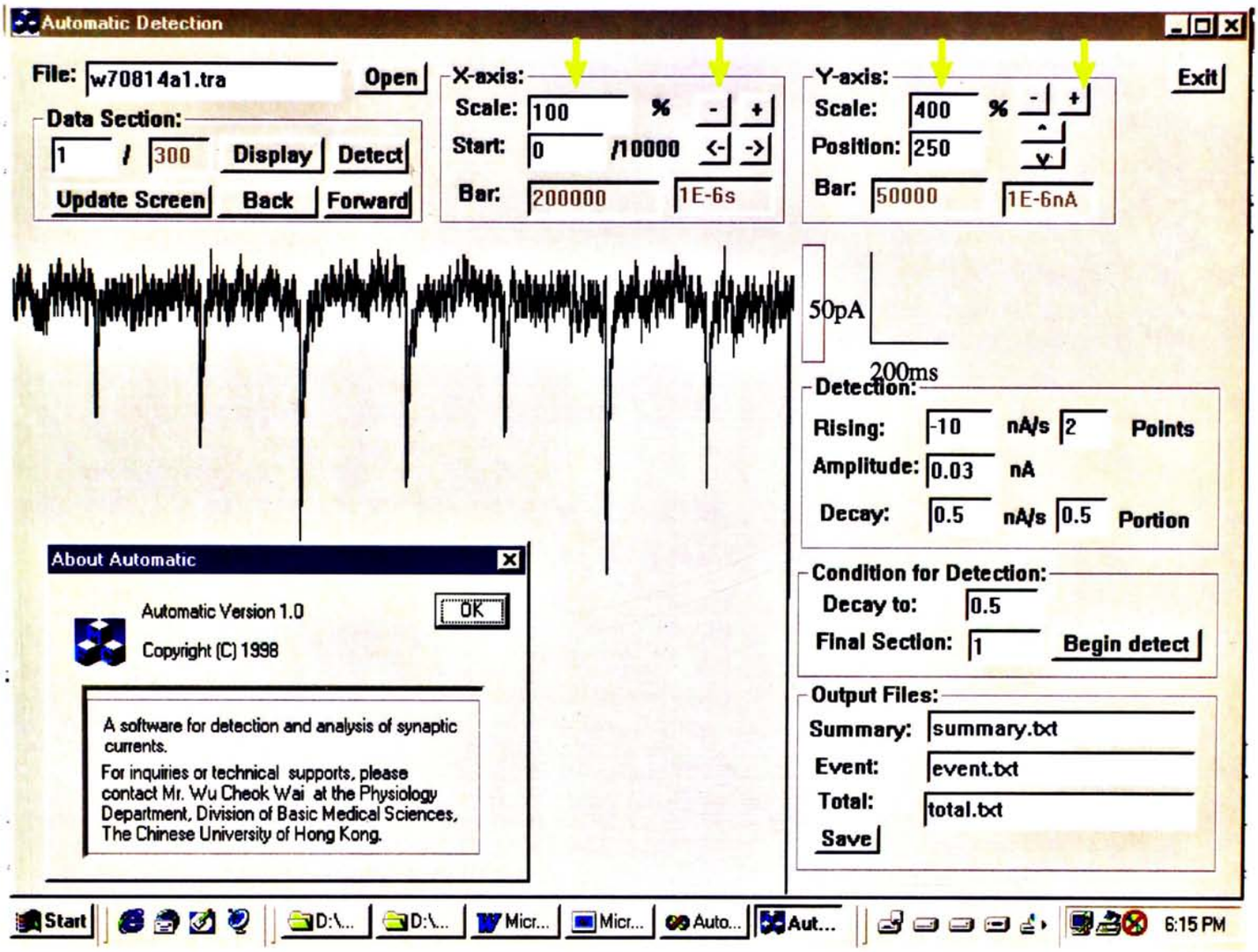


Fig. 3.1. The interface of the Automatic Detection software. It is run under Windows 95. The software displays the recorded $GABA_A$ receptor mediated IPSCs. The software contains buttons and edit boxes that can be used to change the position and scale of the data trace (the yellow arrows). An about dialog box is also displayed to introduce the software.

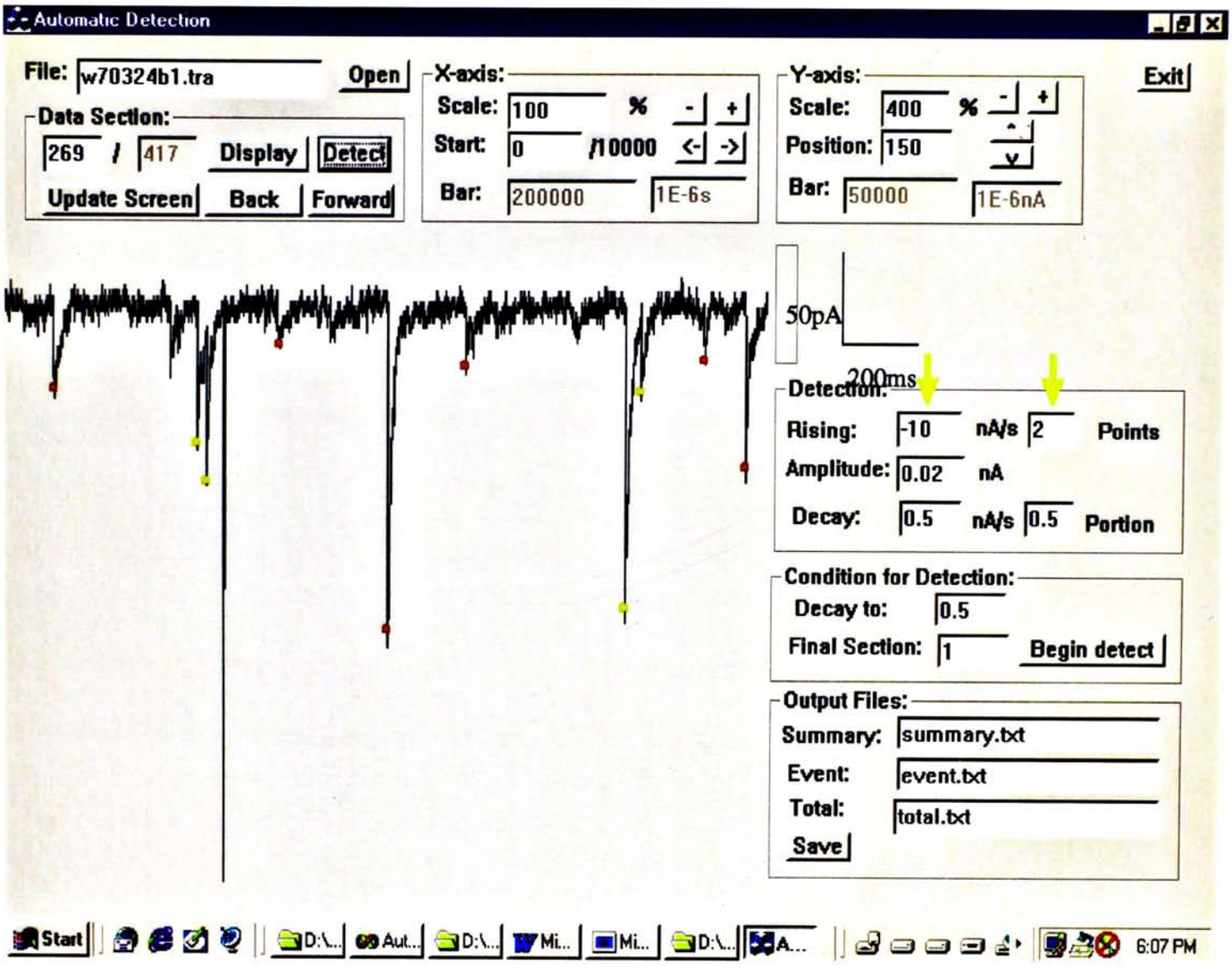


Fig. 3.2. The detected IPSCs. The software contains edit boxes that can be used to set the parameters for detection (the yellow arrows). The detected IPSCs are marked by circles. The red circles represent non-overlapping events while the orange circles represent overlapping events.

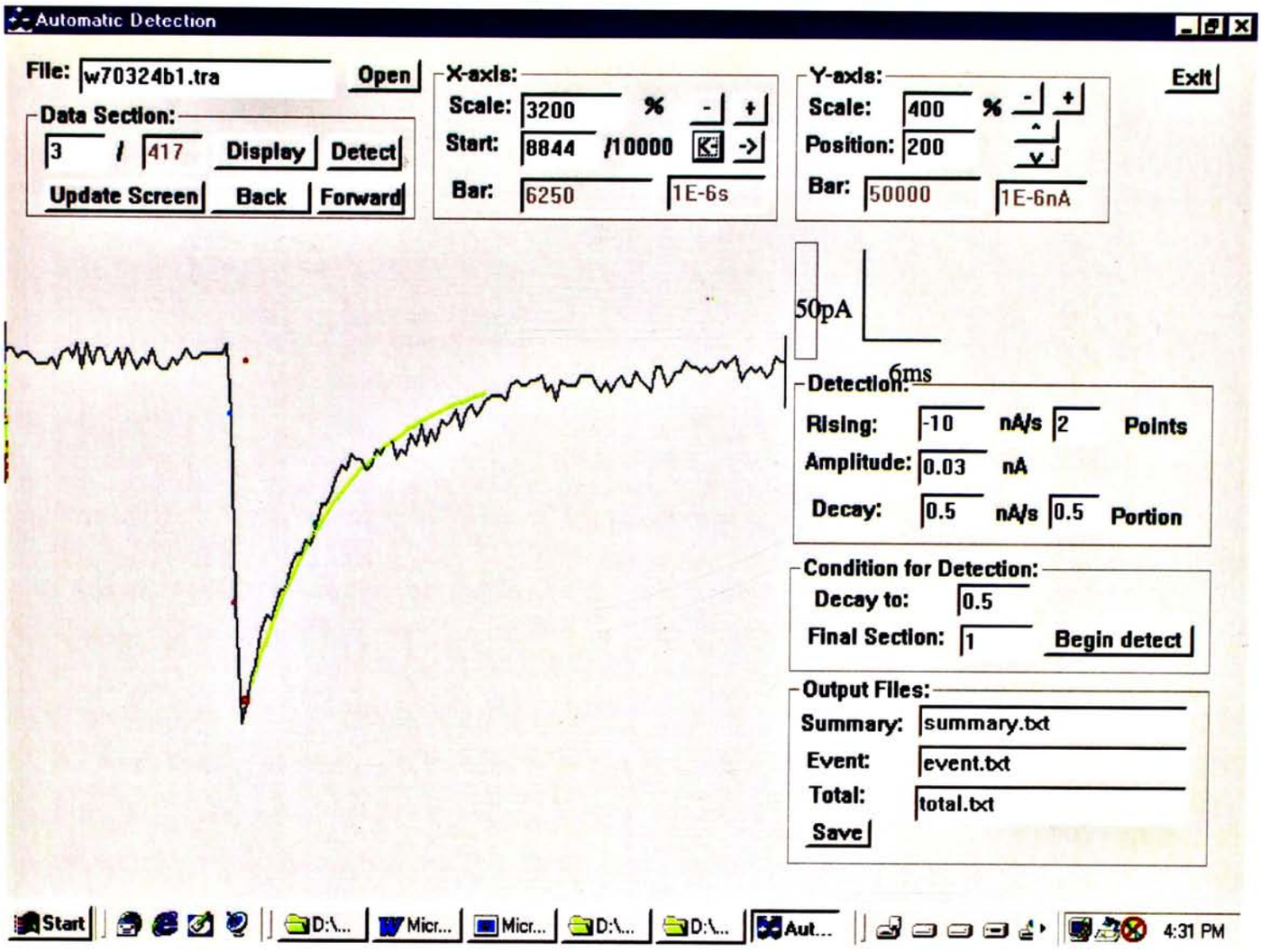


Fig. 3.3. After the events are detected, the software displays graphically the measured base line current, 20% rise time, 80% rise time and the half decay time. The base line line current is displayed as a small red dot, the 20% rise time a cyan dot, the 80% rise time a pink dot and the half decay time a green dot. The large red dot represents the peak of the IPSC. The software also displays the fitting results of the decay phase with the single exponential function (yellow line). As the rising phase of the IPSC is very short, accuracy of location of the 20% rise time and the 80% rise time also depends on the sampling rate.

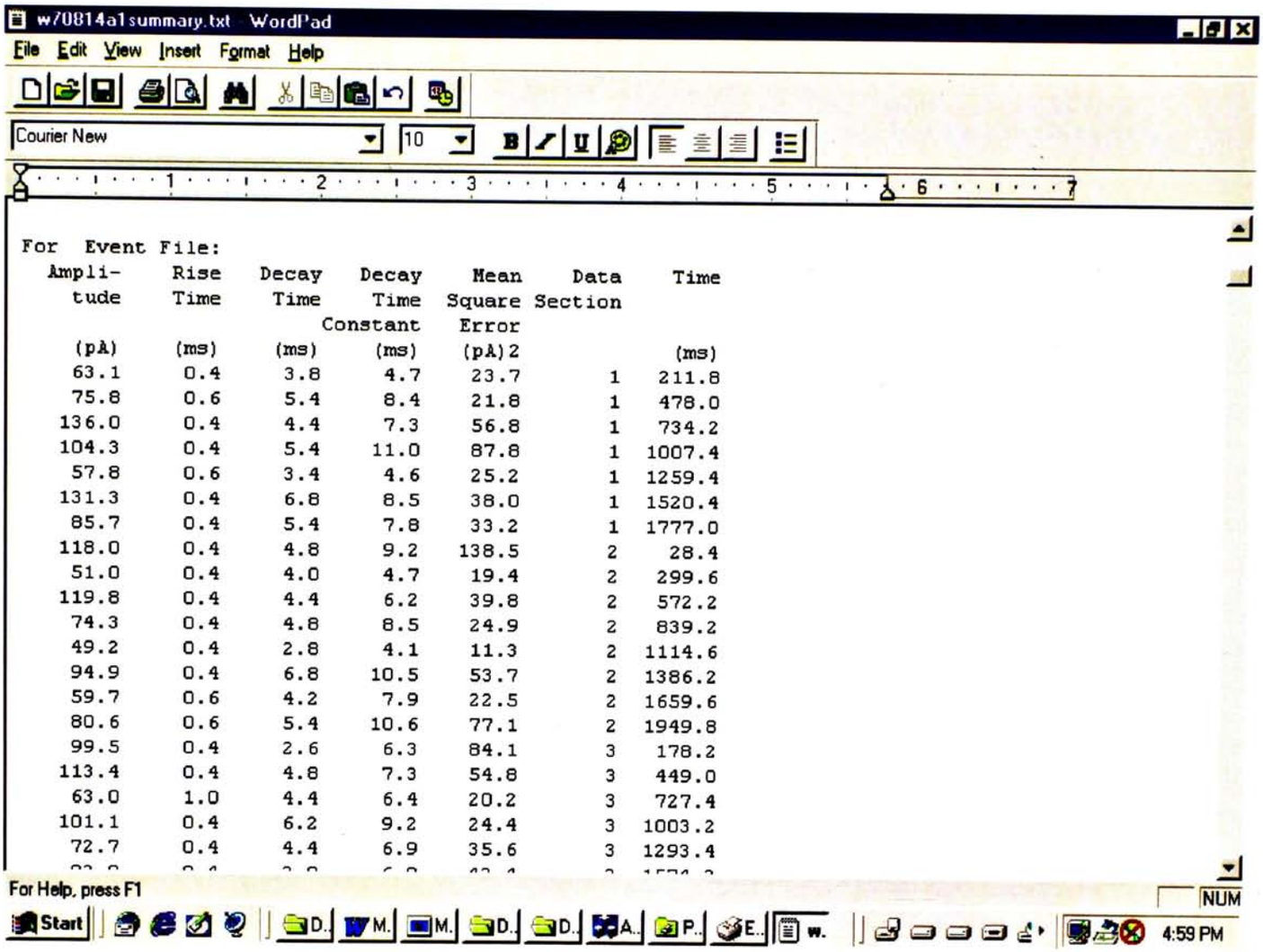


Fig. 3.4. An output text file gernerated after detection. The software gernerates text files that summarize the detction results and the measured parameters of the events. These text files can be imported into other softwares for statistics and gerneration of graphics.

3.2 Performance of the Software

Detection of the spontaneous IPSCs is basically a problem of pattern recognition. When the amplitude of an IPSC is small and comparable with the background noise, the IPSC becomes difficult to be recognized. Different subjects may have different conclusion on whether the signal is a real event or simply a noise. Therefore, even the best detection program will not have 100% accuracy as there is no absolute truth. There are obviously different ways to objectively evaluate the performance of pattern recognition. Since the human can be regarded as the most accurate pattern recogniser, our approach is to measure the consistency of the program when compared to human. However, before evaluating the performance of the program, the consistency of the results of the manual detection method performed by different subjects must be obtained.

Two subjects, subject A and B, were asked to independently detect randomly chosen IPSCs of 4 neurons manually. Data traces with total length equaling to 80000 ms were visually screened. The results obtained by the two subjects were then compared. Subject A reported 383 IPSCs while subject B reported 287 in the 80000 ms data trace. The number of IPSCs exist in both the sets of IPSCs reported by subject A and subject B are 284 (Fig. 3.5).

To quantitatively describe the consistency, an index called the percentage of consistency, or P_{con}, was defined as shown in formula 3.1:

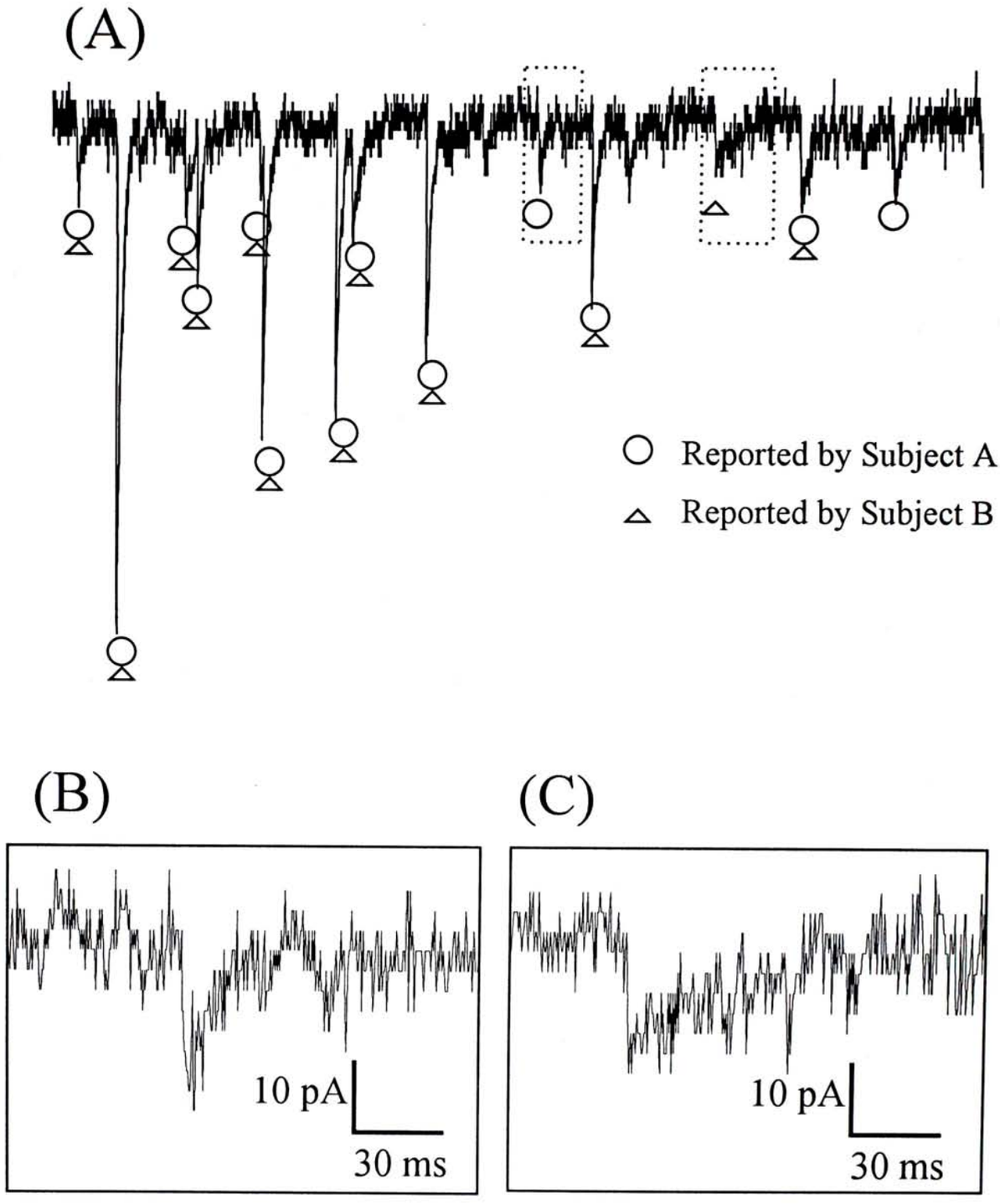


Fig. 3.5. (A) The IPSCs reported by subject A (circle) and B (triangle). The IPSCs with amplitude much larger than the background noise were reported by both subjects A and B. When the amplitude of an 'IPSC' was comparable with the background, it was difficult to make conclusion on whether it was an event or not. Different subjects had different conclusion. (B) Enlargement of the IPSC reported by subject A but not subject B. (C) Enlargement of the IPSC reported by subject B but not subject A.

$$P_{con} = \frac{n}{(N_A + N_B)/2} \times 100\% \qquad (3.1)$$

where N_A and N_B are the number of events reported by subject A and subject B respectively. n is the number of events reported by both subjects. $(N_A + N_B)/2$ represents the mean number of reported events. In our test, the P_{con} between subjects A and B in detecting the same IPSCs is 84.8%. This clearly illustrates that even the best detection method, the manual method, cannot achieve 100% agreement. Obviously, there is no fixed value of P_{con}. The more human subjects involved in the test, the more accurate the mean P_{con} in reflecting consistency among different individuals.

To test the reliability of the Automatic Detection software, the program was used to analyse the same 80000 ms data trace. The user of the program was blind to the results reported by subjects A and B. This prevented any bias in the user. The results were then compared with those obtained by subjects A and B.

With the default settings of the detection variables, the Automatic Detection software detected 306 IPSCs (Fig. 3.6). The number of IPSCs detected by both the program and subject A was 302. This gives a P_{con} value of 87.7%. Similarly, the number of common IPSCs detected by the program and subject B was 266. This gives a P_{con} value of 89.7%. The mean P_{con} value for these two comparison was 88.7% (Fig. 3.7). The common IPSCs reported by both the software and the manual detection methods have large amplitude. Most discrepancies between the software and the manual detection method occurred when the amplitude of the IPSCs becomes comparable with the background

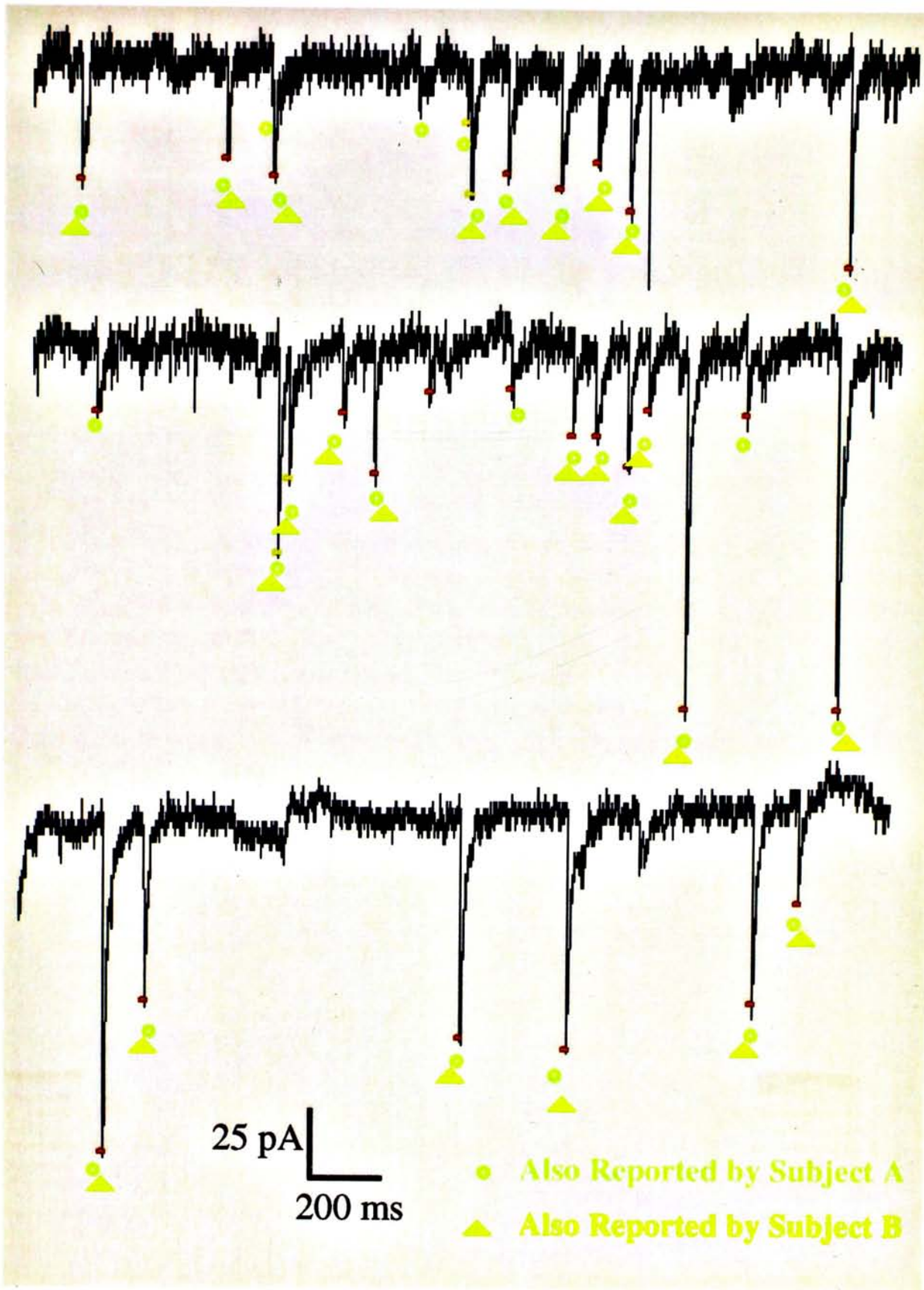


Fig. 3.6. The IPSCs reported by the Automatic Detection software. The software achieved a similar performance as the manual detection method. All the large amplitude IPSCs which were reported by both subjects A and B were also detected by the software. Discrepancy between the software and the manual detection method occured only when the amplitudes of IPSCs were small.

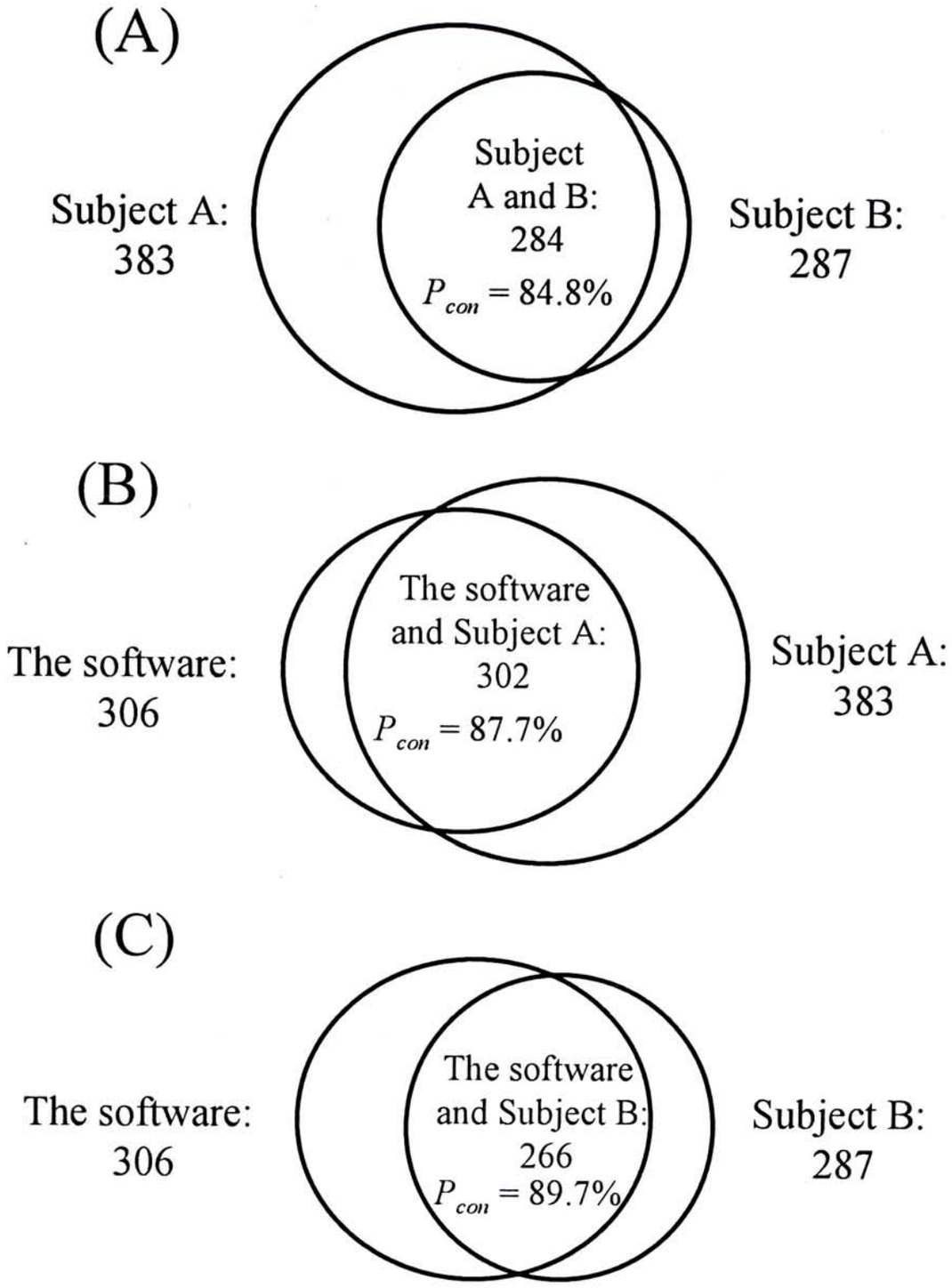


Fig. 3.7a. Set diagrams showing the number of IPSCs reported by subject A, subject B and the Automatic Detection software. The P_{con} value between subject A and B was 84.8%. The P_{con} value between the software and subject A was 87.7%. The P_{con} value between the software and subject B was 89.7%. The software had performance comparable with the manual detection method.

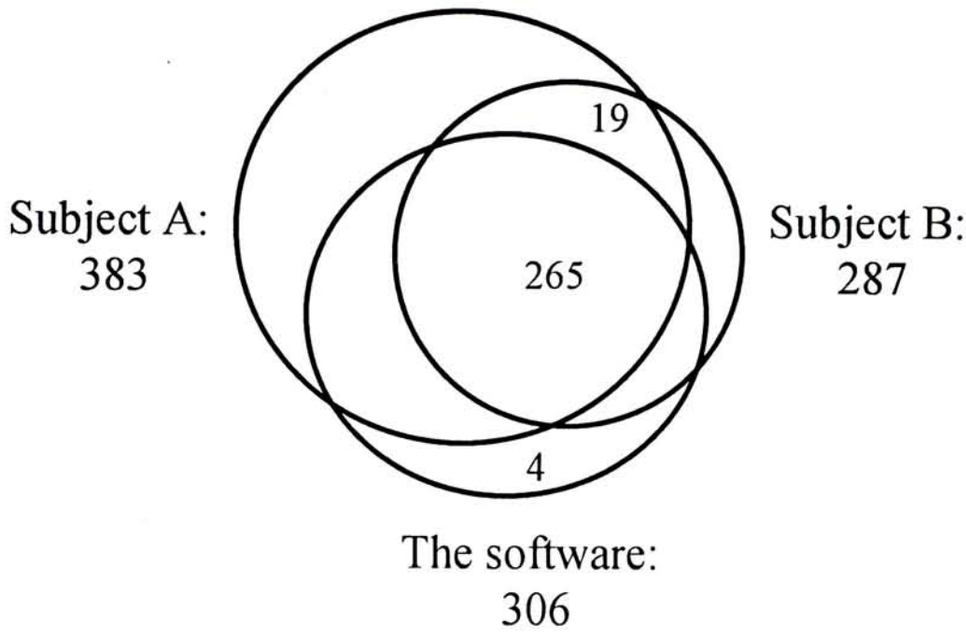


Fig. 3.7b. Among those IPSCs reported by both subject A and B, 265 IPSCs were also reported by the Automatic Detection software. These IPSCs were mainly large amplitude IPSCs. Only 19 IPSCs which were reported by both subject A and B were not reported by the Automatic Detection software. The Automatic Detection software reported 4 IPSCs which were neither reported by subject A nor by subject B. These IPSCs were mainly small amplitude IPSCs (see also Fig. 3.6).

noise (Fig. 3.6). As the mean P_{con} value between the Automatic Detection software and the manual detection method (88.7%) was comparable with the P_{con} value between the subject A and B (84.8%), the Automatic Detection software had performance similar to a human subject.

Within these 80000 ms data trace, 8 fast noises were visually identified by both subjects. None of these were labeled by the software as an event of IPSC during the detection process (Fig. 3.8). Among those 306 IPSCs detected by the software, 48 IPSCs were visually examined by one human subject to be overlapping events while the program reported 49 overlapping IPSCs. All the 48 visually determined overlapping IPSCs were also reported by the software (Fig. 3.9).

3.3 Discussion

As discussed in the last section, when the signal to noise ratio is small, different subjects may have different conclusions on whether a signal is an IPSC. We cannot say which subject's conclusion is right. Instead, we can only say how consistent the two subjects are. To quantitatively describe this consistency, the P_{con} value was defined in formula 3.1. Its definition is meaningful. First, its value is always between 0 and 1. If two subjects do not report any common IPSCs, n will equal to 0 and P_{con} will be 0. That means the conclusions made by the two subjects are not consistent at all. If the two subjects report the same pool of IPSCs, $n = N_A = N_B$ and $P_{con} = 100\%$. That means the conclusions made by the two subjects are totally consistent. Second, if subject A reports 100 IPSCs, subject B reports 100 IPSCs and there are 50

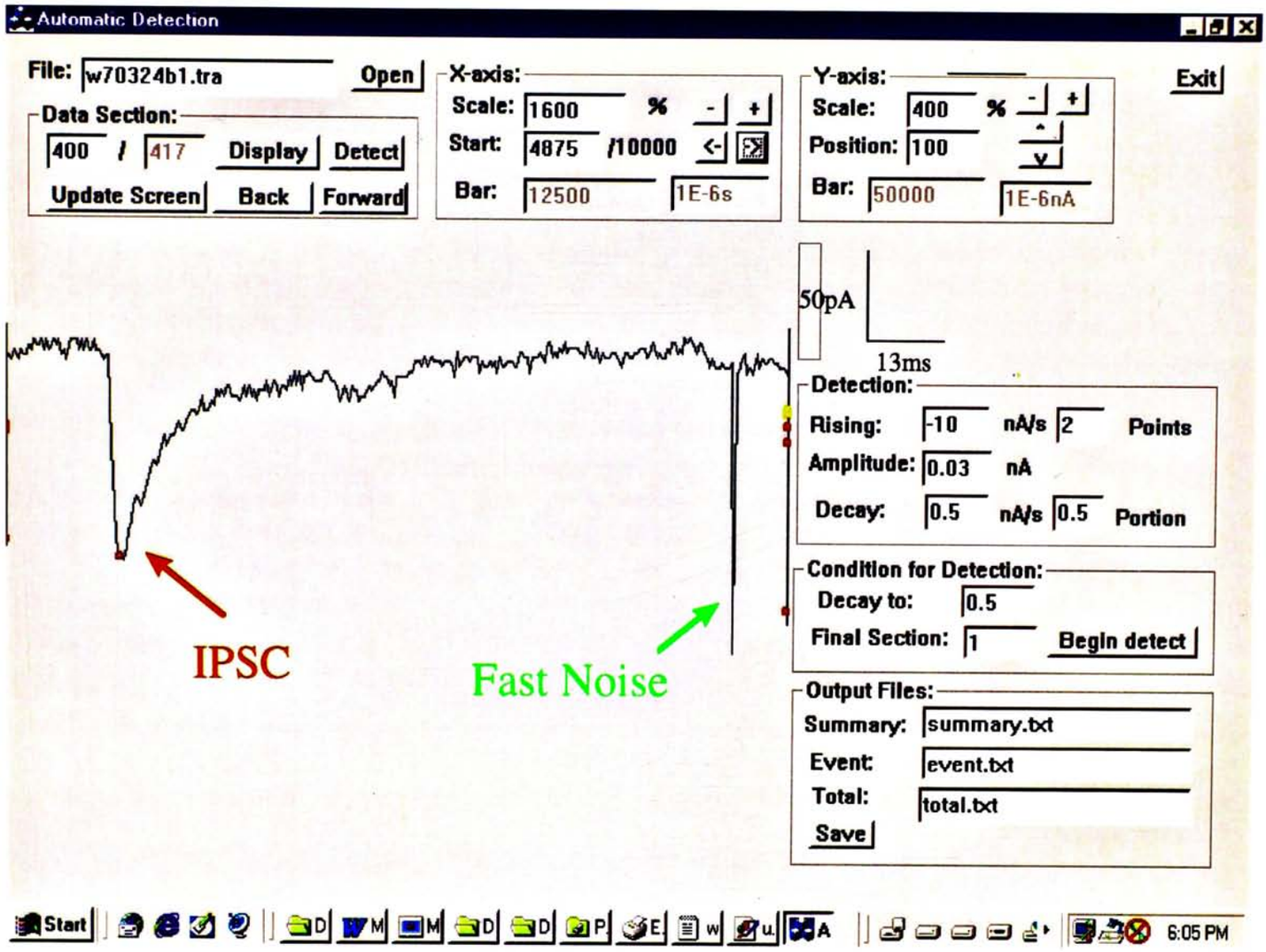


Fig. 3.8. The Automatic Detection software successfully distinguishes events of IPSCs and fast noise transient. In the above diagram, although the fast noise has larger amplitude than the IPSC, the software only reports the IPSC as an event instead of the fast noise. As the scale of the middle part data trace is enlarged, the data trace at the two ends are compressed accordingly. The circles at the two ends of the data trace represent events of IPSCs in these parts of the data trace.

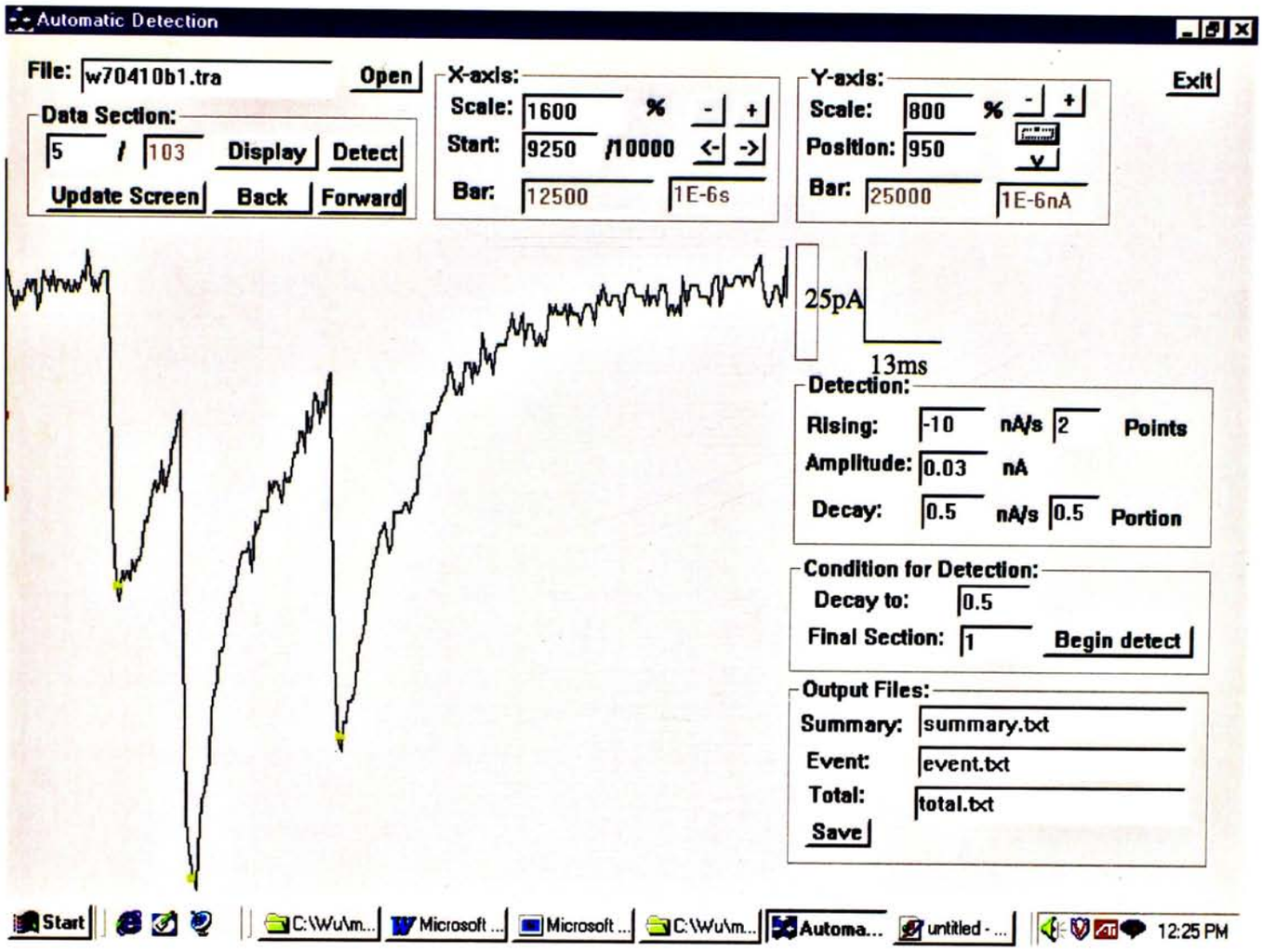


Fig. 3.9. The Automatic Detection software is also successful in detecting the overlapping events of IPSCs. In the above figure, all the three events are detected by the software. The software also correctly classified them as overlapping events and labelled them with orange circles.

common IPSCs, n = 50, N_A = 100 and N_B = 100, and P_{con} = 50%. That means a half consistency between the two subjects.

Those IPSCs reported by both the software and the manual detection have large amplitudes. Discrepancy occurred only when the amplitudes of the IPSCs were comparable with the background noise. Moreover, as the mean P_{con} value between the program and the manual detection method (88.7%) was comparable with the P_{con} value between subject A and subject B (84.8%), the degree of consistency arrived by the software and a human subject is comparable to those made by two human subjects. Therefore, according to this definition, the performance of the software is comparable to that of a human subject.

A common artifact in electrophysiological recordings is the fast transient noise. Our program addresses this issue by implementing a specific fast noise test. When compared with an algorithm that simply relies on test of a threshold amplitude, this software should be more robust in rejecting fast noises. This is indicated by the fact that in our test, there was not a single fast noise, judged by a human subject, that was labeled by the software as an IPSC. The program also helps to improve the accuracy of the kinetic data of the IPSCs by detecting the overlapping events which were not used in calculating the mean data. This problem is not addressed by the scaled template method (Clements and Bekkers, 1997).

In conclusion, the program developed in the present project performs satisfactorily in detecting spontaneous $GABA_A$ mediated IPSCs. However, in

the further development of the software, many things can still be done. The software may incorporate other functions such as plotting statistical graphs and performing statistical tests to make the software more versatile. More analysis of the software may also be added to fully evaluate its performance.

Chapter 4: Application

4.1 Introduction

This chapter describes an application example of the Automatic Detection software on the detection, and measurement of parameters, of IPSCs at an identified synapse: the GABA inhibitory synapses of substantia nigra dopamine neurons. This serves as an example of how the software can help us to address real biological question.

The dopaminergic neurons of the substantia nigra pars compacta (SNC) are very important in the function of controlling the motor activity of the body, as evidenced by the resulting Parkinson's disease after their degeneration. As the activity of the dopaminergic neurons are mainly controlled by inhibition through GABAergic projections originating from the striatum, globuas pallidus and within the substantia nigra itself (Smith and Bolam, 1990; Nicholson *et al.*,1992; Hajos and Greenfield, 1993; Tepper, 1995), the kinetic data of the $GABA_A$ mediated IPSCs will provide us the information of the process of GABA action itself, and how the activity of the dopaminergic neurons might be modified as therapeutic targets. The properties of GABA mediated postsynaptic potentials (IPSPs) have been studied in the brain by Häusser and Yung (1994). However, surprisingly, there is little information on the basic characteristics of the underlying IPSCs.

In this study, GABA mediated IPSCs in SNC were recorded from SNC by the whole-cell patch-clamp method. The Automatic Detection program was then used to analyse their characteristic. In addition, the role of GABA uptake in controlling the decay kinetcis of GABA on SNC dopamine neurons was studied. The characteristics of the IPSCs before and after the application of NO-711 Hydrochloride was compared.

4.2 Materials and Methods

Brain Slice Preparation

Sparague-Dawley rats of 14 days old were decapitated. The brains were quickly removed and immersed in ice-cold artificial cerebrospinal fluid (ASCF) with composition described in Table 4.1. Coronal midbrain slices in 350 μm containing the substantia nigra were cut with a vibratome (Fig. 4.1). The brain slices were kept in a holding chamber with oxygenated ACSF at 35°C for at least 30 minutes for equilibrium.

Composition	Concentration (mM)
NaCl	120
KCl	2
$MgSO_4$	1.2
KH_2PO_4	1.2
$NaHCO_3$	26
$CaCl_2$	2.5
Glucose	11

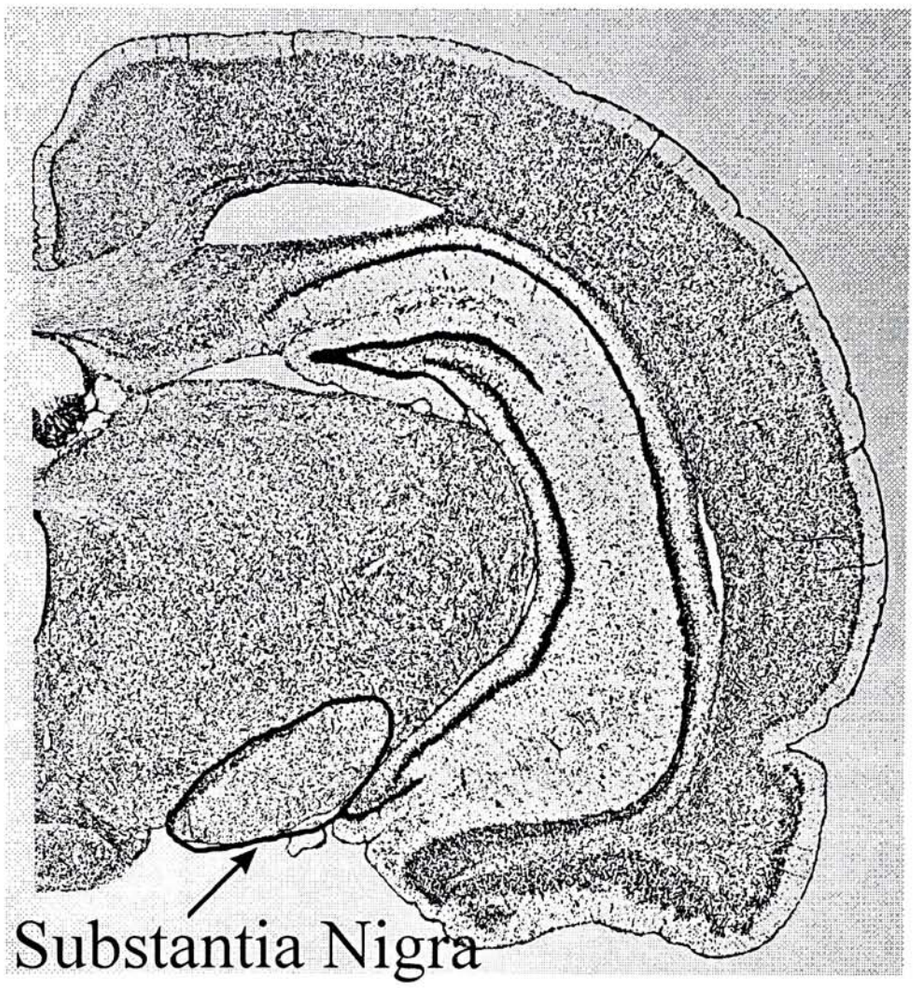


Fig. 4.1. To obtain the required brain slices , the rat brains were cut coronally using a vibratome caudal - rostrally. Brain slices in 350μm containing the substantia nigra were taken. The whole cutting process was performed in a chamber filled with ice-cold ACSF. (The diagram was taken from Swanson, 1992).

Table 4.1 Composition of the ASCF.

Patch Clamp Recording

A brain slice was transferred into a recording chamber which was superfused with oxygenated ACSF at 35±2°C at a flow rate of 1.5-2.0 ml per minute. For patch-clamp recording, the neurons within the brain slice was visually identified with a upright microscope (Zeiss Axioskop). Optical sectioning of the brain slice can be obtained using the differential interference contrast optics of the microscope (Fig. 4.2). The associated infra-red video camera and related electronic circuit improved the contrast between the neurons and the background structure. (Stuart *et al.*, 1993).

Borosilicate glass capillaries were pulled with a Flaming/Brown Micropipette Puller, Model P-87, (Sutter Instrument) to produce the patch pipettes. Intracellular solution was prepared using the composition described in table 4.2. It was titrated to pH 7.3 with KOH.

Composition	Concentration (mM)
KCl	130
HEPES	10
EGTA	1
$MgCl_2$	2
Na_2ATP	2

Table 4.2 Composition of the intracellular solution.

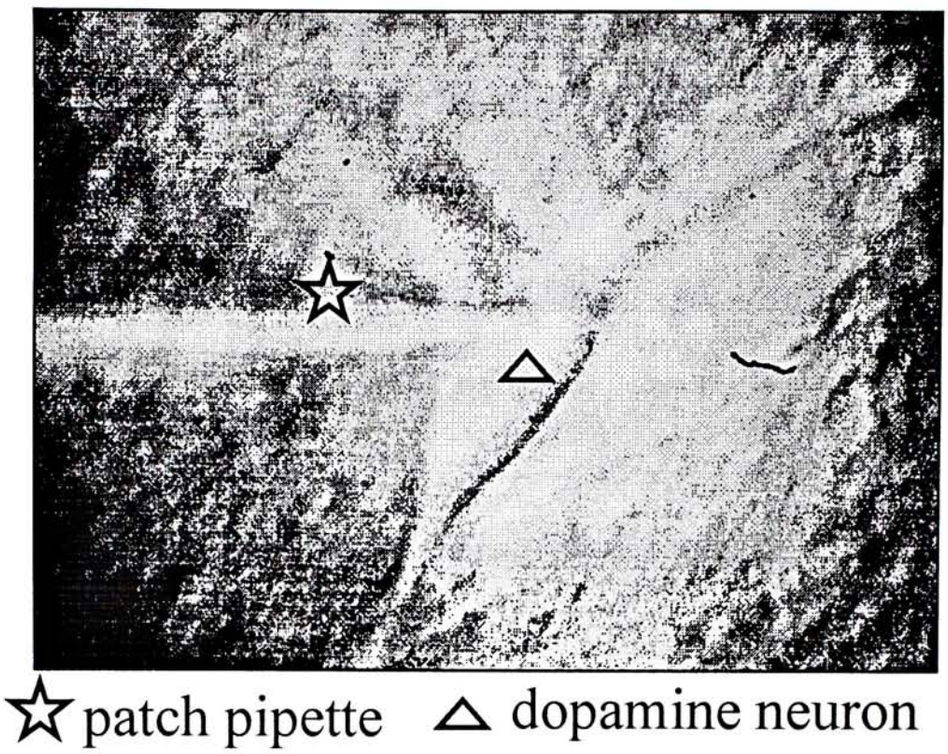


Fig. 4.2. A patched neuron identified under a microcope. The interference lenses within the microscope and the infra-red camera together provide a clear contrast between the neuron and the background structure.

To patch a neuron, positive pressure of 60-120 mbar was applied through the patch pipette to clear the membrane surface. Then small negative pressure, 10-20 mbar, was applied to the pipettes to form gigaohm seals with the neurons. A more negative pressure was than applied to the pipettes to break through the cell membrane (Hamill *et al.*, 1981) to achieve whole-cell configuration.

Recordings were made using a patch clamp amplifier (LM/PCA, List Medical) controlled by the CED Voltage and Patch Clamp software (Version 6.0, Cambridge Electronics Design). Data were filtered at 3 kHz and stored with Sony digital audio tape. A chart recorder was also used to display the current. The data were also on-line sampled at 2.5-5.0 kHz and stored into a computer disk via the CED software. The data were analyzed off-line with the Automatic Detection software, details of this have been described in the previous chapters.

The electrophysiological characteristics of the patched neurons is studied to confirmed that they were dopaminergic neurons. Under current clamp mode, steps of large hyperpolarizing current pulses were injected into the neurons. This would induce a characteristic time dependent hyperpolarization activated inward rectification in the dopaminergic neurons. (Yung *et al.*, 1991) (Fig. 4.3).

Data were presented as mean±SE. Tests were performed by paired Student's t-test. Drugs were added to the ACSF through the perfusion system to the brain slices. Bicuculline and NO-711 were obtained from Research Biochem International.

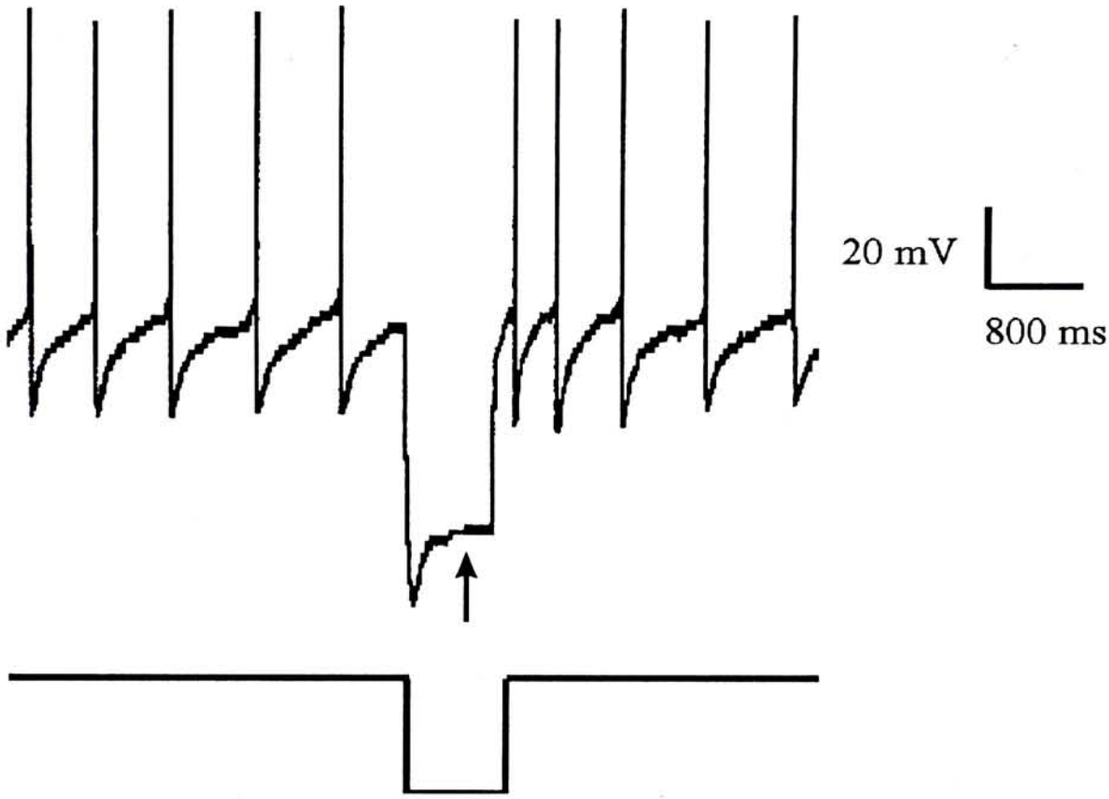


Fig. 4.3. The dopaminergic neurons displayed a regular firing pattern. When a hyperpolarising current pulse was injected into the neurons, the neurons displayed time dependent hyperpolarization activated inward rectifications (the arrow). The rectification was mediated by an increase in the membrane conductance to Na^+ and K^+.

4.3 Result

The neurons were held at -70 mV to observe and record the postaynaptic synaptic currents. The IPSCs of the dopaminergic neurons appeared randomly with variable amplitudes, 20-80% rise times and decay time constants. The frequency also varied from cell to cell. It ranged from 0.03 to 6.10 Hz (n=38). The mean frequency of the synaptic currents was 1.64±0.27 Hz. The synaptic currents can be blocked by 1μM bicuculline (n=4) (Fig. 4.4). This confirmed that synaptic currents are mediated by the $GABA_A$ receptors.

To characterize the IPSCs of each neuron, the Automatic Detection software was use to measure the amplitude, 20-80% rise time and the decay time constant of the IPSCs (Fig. 4.5). Then the mean values of these parameters of several hundred IPSCs of each neurons were calculated. The mean amplitude, 20-80% rise time and decay time constant are 59.68±4.15 pA, 1.04±0.06 ms and 10.67±0.51 ms respectively (38 cells).

Addition of 100 μM NO-711 Hydrochloride decreased the mean frequency of the IPSCs from 1.64±0.39 Hz to 0.55±0.16 Hz (9 cells, $P<0.005$) (Fig. 4.6). NO-711 Hydrochloride also increased the decay time constant from 9.30±0.53 ms to 10.73±0.66 ms (9 cells, $P<0.001$). The 20-80% rise time of the neurons increased from 0.83±0.06 ms to 1.27±0.05 ms (9 cells, $P<0.005$). The amplitude of the IPSCs also decreased form 62.62±4.76 pA to 51.94±4.49 pA (9 cells, $P<0.05$).

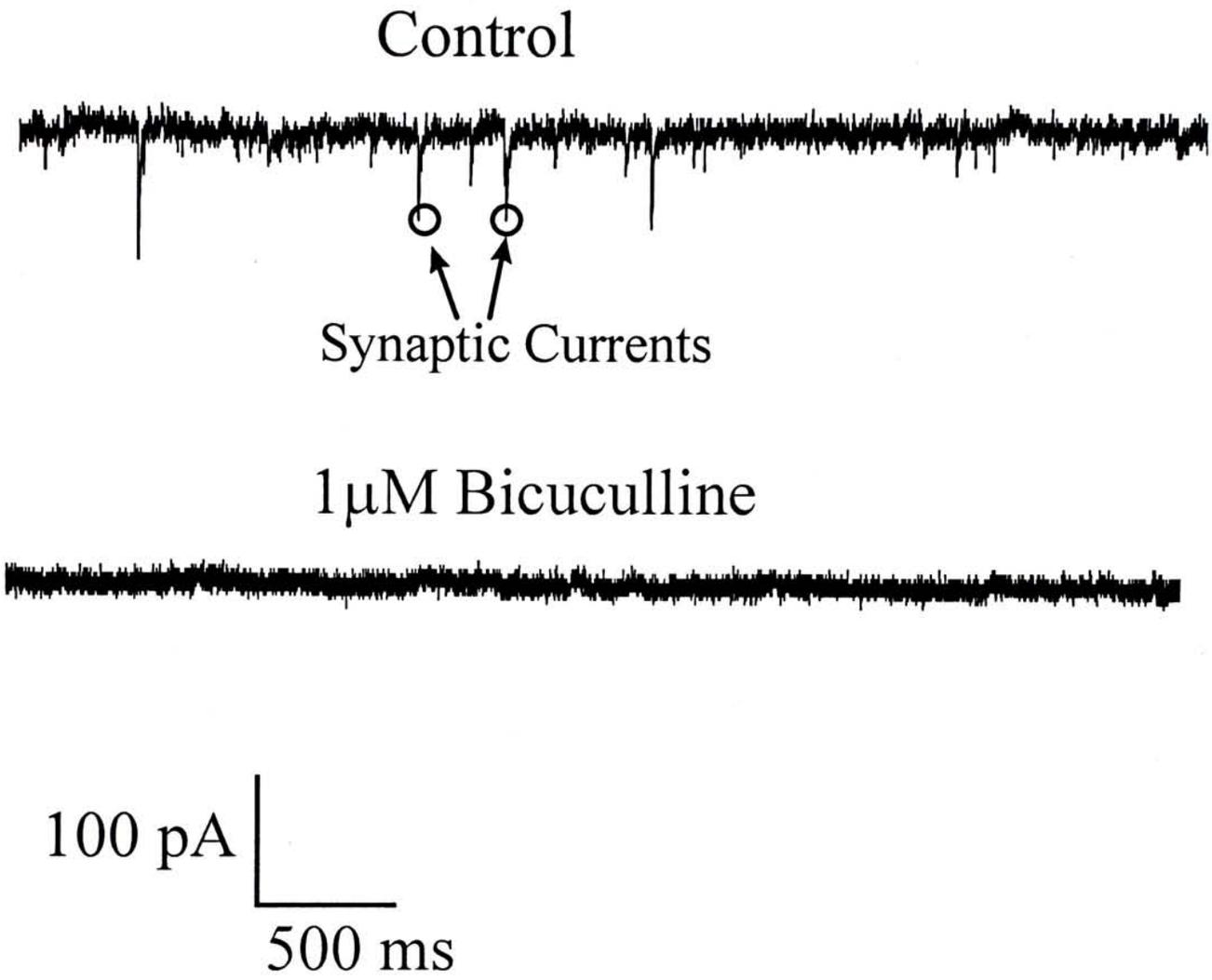


Fig. 4.4. The synaptic currents of the dopaminergic neurons appeared as downward deflections from the base line. After adding 1μM bicuculline, the $GABA_A$ receptor antagonist, the synaptic currents disppeared. This confirmed that they were mediated by the $GBAB_A$ receptors.

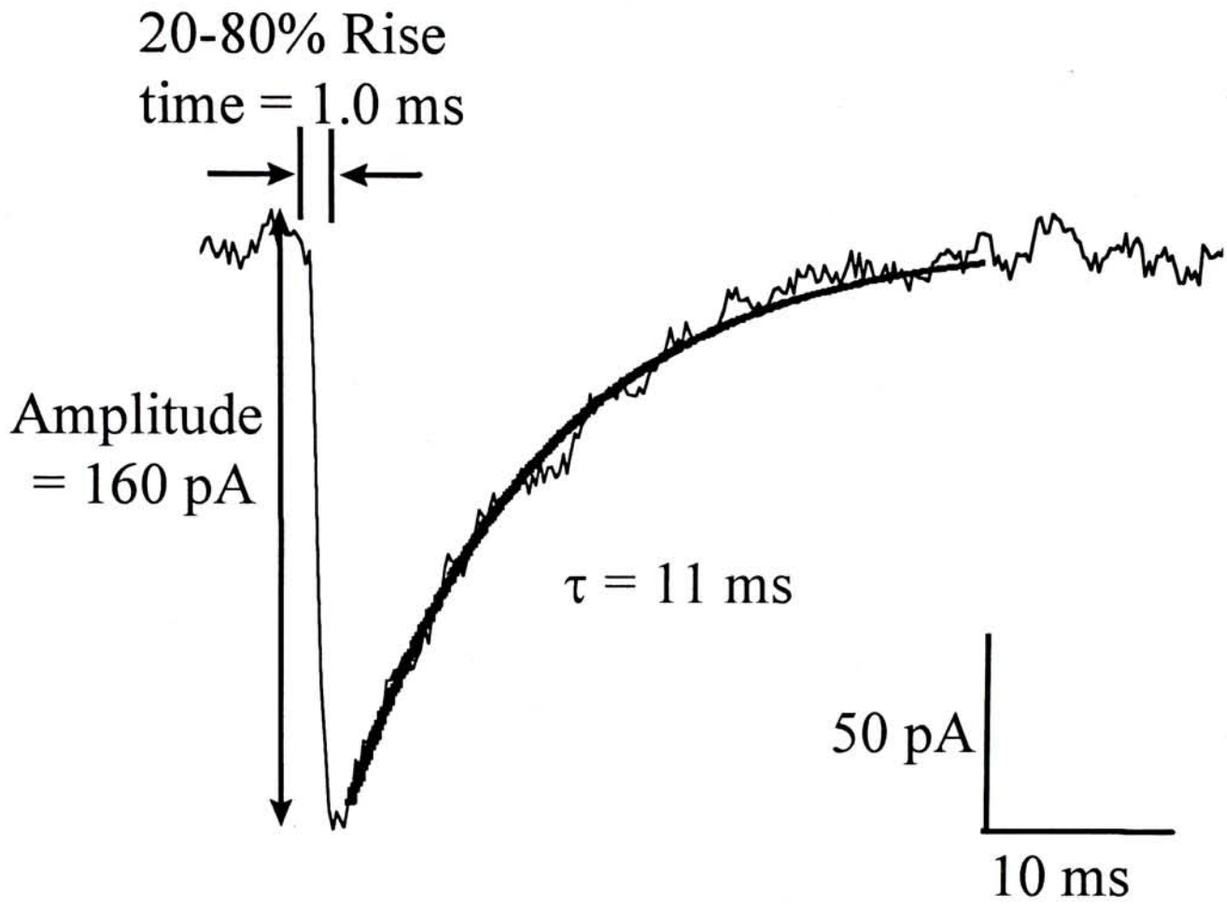


Fig. 4.5. The kinetics of an explaned IPSC. To characterise the IPSCs of the neurons, the 20-80% rise time and the amplitude were measured. The decay phase of the IPSCs was also fitted with a single exponential function to find out the decay time constant. In this example, the 20-80% rise time was 1.0 ms, the amplitude was 160 pA and the decay time constant was 11 ms. For each neuron, several hundred IPSCs are characterzed to calculate the mean value of the parameters of the IPSCs.

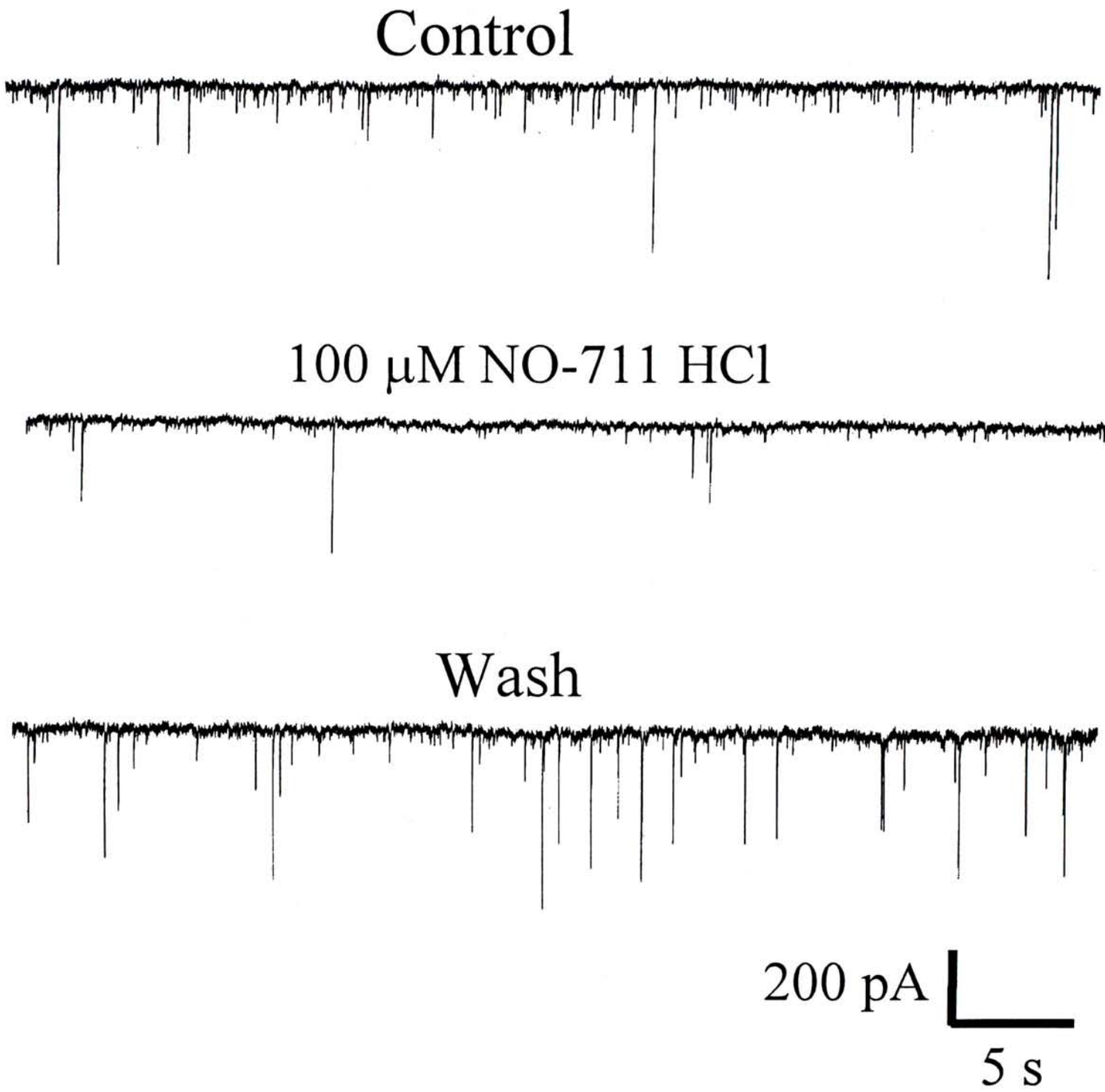


Fig. 4.6. Effect of NO-711 Hydrochloride on the IPSCs of the dopaminergic neurons. NO-711 Hydrochloride reduced the frequency of the IPSCs. The amplitudes also tended to decrease. After washing, the frequency was restored.

4.4 Discussion

We have used 2.5-5.0 kHz sampling rate to collect and store the data of the IPSCs into a computer disk for off-line data analysis. As the sampling rate was relatively low, the measured 20-80% rise time of the IPSCs was not very accurate. The rise time only had a resolution of 0.2-0.4 ms depending on the sampling rate used. As the rise time of the IPSCs was in the range of 0.6-2.0 ms, the percentage error was relatively high for each individual IPSC. Higher sampling rate should be used to obtain a more accurate result.

NO-711 Hydrochloride caused a significant increase in the decay time constant of the IPSCs. The result suggests that GABA uptake plays an important role in regulating the kinetics of the $GABA_A$ mediated IPSCs. Blocking the GABA uptake system decreases the rate of removal of GABA from the synaptic clefts and increases the concentration of GABA in this region.

NO-711 Hydrochloride also caused a significant decrease in the frequency of the $GABA_A$ receptors mediated IPSCs. This may be explained by the hypothesis that as the concentration of GABA in the synaptic clefts is increased, the $GABA_B$ autoreceptors located in the presynaptic terminals are activated. The activated $GABA_B$ receptors then reduce the frequency of those miniature IPSCs from the presynaptic terminals and so the overall frequency of the IPSCs recorded decreased. This also explains why the amplitude of the IPSCs decrease. As the presynaptic $GABA_B$ autoreceptors are activated, the amount of GABA released each time decreased and so the amplitude of the IPSCs decrease. The $GABA_B$ autoreceptors located in the presynaptic neurons

may provide a feedback mechanism that prevents over inhibition to the dopaminergic neurons.

In the hippocampal slice preparation, tiagabine increased the half width of the evoked IPSCs of the neurons while it had no significant change on the amplitude (Roepstorff and Lambert, 1992). In cultural hippocampal neurons, NO-711 Hydrochloride did not change the decay time constant of spontaneous IPSCs but decreased their amplitude (Oh and Dichter, 1994). Therefore, it seems that the net effect of GABA uptake inhibitors depends on the particular environment of the neurons. Different extracellular space and glial cells combination may cause different effects of the GABA uptake inhibitors on the amplitude and the decay time constant of the IPSCs.

Reference

Andrade R, Malenka RC and Nicoll RA (1986). A G protein couples serotonin and $GABA_B$ receptors to the same channels in hippocampus. *Science* **234**, 1261-1265.

Andrews M (1996).C++ Basics. In *Learn Visual C++ Now*. ed. Clark D, Harris J, Anagnost C and Pierce J. pp. 49-82. Microft Press, USA.

Ankri N, Legendre P, Faber DS and Korn H (1994). Automatic detection of spontaneous synaptic responses in central neurons. *J. Neurosci. Methods* **52**, 87-100.

Barker JL, McBurney RN (1979). Phenobarbitone modulation of postsynaptic GABA receptor function on cultured mammalian neurons. *Proc. R. Soc. Lond. B. Biol. Sci.* **206**, 319-327.

Bier M Kits KS and Borst JG (1996). Relation between rise times and amplitudes of GABAergic postsynaptic currents. *J. Neurophysiol.***75**, 1008-1012.

Borowsky B and Hoffman BJ (1995). Neurotransmitter transporters: molecular biology, function, and regulation. *Int. Rev. Neurobiol.* **38**, 139-199.

Clements JD and Bekkers JM (1997). Detection of spontaneous synaptic events with an optimally scaled template. *Biophys. J.* **73**, 220-229.

Cochran SL (1993). Algorithms for detection and measurement of spontaneous events. *J. Neurosci. Methods* **50**, 105-121.

Curtis DR, Duggan AW, Felix D, Johnston GA and McLennan H (1971). Antagonism between bicuculline and GABA in the cat brain. *Brain Res.* **33**, 57-73.

Del Castillo J and Katz B (1954) Quantal components of the end-plate potential. *J. Physiol. Lond.*, **124**, 560-573.

Dempster J (1993) An introduction to computer. In *Computer Analysis of Electrophysiological Signals*, ed. Sattelle DB. pp. 204-213. Academic Press, Londan.

Dingledine R and Gjerstad L (1980). Reduced inhibition during epileptiform activity in the in vitro hippocampal slice. *J. Physiol. (Lond)* **305**, 297-313.

Fonnum F and Walberg F (1973). An estimation of the concentration of γ-aminobutyric acid and glutamate decarboxylase in the inhibitory Purkinje axon terminals in the cat. *Brain Res.* **54**, 115-127.

Franaszczuk PJ, Bergey GK and Kudela P (1995). Detection of spontaneous postsynaptic potentials. *Comput. Biomed. Res.***28**, 354-370.

Fromm GH, Sato K and Nakata M (1992). The action of $GABA_B$ antagonists in the trigeminal nucleus of the rat. *Neuropharmacology* **31**, 475-480.

Gurewich O and Gurewich N (1997). Learn C/C++ quiockly (Part I). In *Teach Yourself Visual C++ 5 in 21 Days.* ed. Morrow C. pp. 665-708. SAMS Publishing, USA.

Hajos M and Greenfield SA (1993). Topographic heterogeneity of substantia nigra neurons: diversity in intrinsic membrane properties and synaptic inputs. *Neuroscience* **55**, 919-934.

Hamill OP, Marty A, Neher E, Sakmann B and Sigworth FJ (1981). Improved patch-clamp techniques for high-resolution current recording from cells and cell-free membrane patches. *Pflugers. Arch.* **391**, 85-100.

Hausser MA and Yung WH (1994). Inhibitory synaptic potentials in guinea-pig substantia nigra dopamine neurones *in vitro*. *J. Physiol. (Lond)* **479**, 401-422.

Hill DR and Bowery NG (1981). ^{3}H-baclofen and ^{3}H-GABA bind to bicuculline-insensitive $GABA_B$ sites in rat brain. *Nature* **290**, 149-152.

Llano I and Gerschenfeld HM (1993). Inhibitory synaptic currents in stellate cells of rat cerebellar slices. *J Physiol (Lond)* **468**, 177-200.

Macdonald RL, Olsen RW (1994). $GABA_A$ receptor channels. *Annu. Rev. Neurosci.* **17**, 569-602.

Nicholson LF, Faull RL, Waldvogel HJ and Dragunow M (1992). The regional, cellular and subcellular localization of $GABA_A$/benzodiazepine receptors in the substantia nigra of the rat. *Neuroscience* **50**, 355-370.

Oh DJ and Dichter MA (1994). Effect of a gamma-aminobutyric acid uptake inhibitor, NNC-711, on spontaneous postsynaptic currents in cultured rat hippocampal neurons: implications for antiepileptic drug development. *Epilepsia* **35**, 426-430.

Olsen RW (1981). GABA-benzodiazepine-barbiturate receptor interactions. *J Neurochem* **37**, 1-13.

Otis TS and Mody I (1992). Modulation of decay kinetics and frequency of $GABA_A$ receptor-mediated spontaneous inhibitory postsynaptic currents in hippocampal neurons. *Neuroscience* **49**, 13-32.

Rabow LE, Russek SJ and Farb DH (1995). From ion currents to genomic analysis: recent advances in $GABA_A$ receptor research. *Synapse* **21**, 189-274.

Roepstorff A and Lambert JD (1992). Comparison of the effect of the GABA uptake blockers, tiagabine and nipecotic acid, on inhibitory synaptic efficacy in hippocampal CA1 neurones. *Neurosci. Lett.* **146**, 131-134.

Salin PA and Prince DA (1996). Spontaneous $GABA_A$ receptor-mediated inhibitory currents in adult rat somatosensory cortex. *J. Neurophysiol.* **75**, 1573-1588.

Schildt H (1990). An Overview of C. In *C: The Complete Reference*, pp.3-16 McGraw-Hill,USA.

Schousboe A (1981). Transport and metabolism of glutamate and GABA in neurons are glial cells. *Int. Rev. Neurobiol.* **22**, 1-45.

Smith Y and Bolam JP (1990). The output neurones and the dopaminergic neurones of the substantia nigra receive a GABA-containing input from the globus pallidus in the rat. *J. Comp. Neurol.* **296**, 47-64.

Solis JM and Nicoll RA (1992). Pharmacological characterization of $GABA_B$-mediated responses in the CA1 region of the rat hippocampal slice. *J. Neurosci.* **12**, 3466-3472.

Srinivasan V, Neal MJ and Mitchell JF (1969). The effect of electrical stimulation and high potassium concentrations on the efflux of (^3H) gamma-aminobutyric acid from brain slices. *J. Neurochem.* **16**, 235-1244.

Stelzer A (1992). $GABA_A$ receptors control the excitability of neuronal populations. *Int. Rev. Neurobiol.* **33**, 195-287.

Stuart GJ, Dodt HU and Sakmann B (1993) Patch-clamp recordings from the soma and dendrites of neurons in brain slices using infrared video microscopy. *Plugers Arch.* **423**: 511-518.

Study RE and Barker JL (1981). Diazepam and (--)-pentobarbital: fluctuation analysis reveals different mechanisms for potentiation of gamma-aminobutyric acid responses in cultured central neurons. *Proc. Natl. Acad. Sci. U S A* **78**, 7180-7184

Suzdak PD, Frederiksen K, Andersen KE, Sorensen PO, Knutsen LJ and Nielsen EB (1992). NNC-711, a novel potent and selective gamma-aminobutyric acid uptake inhibitor: pharmacological characterization. *Eur. J. Pharmacol.* **224**, 189-198.

Tepper JM, Martin LP and Anderson DR (1995). $GABA_A$ receptor-mediated inhibition of rat substantia nigra dopaminergic neurons by pars reticulata projection neurons. *J. Neurosci.* **15**, 3092-3103.

Vicini S, Mienville JM and Costa E (1987). Actions of benzodiazepine and beta-carboline derivatives on gamma-aminobutyric acid-activated Cl^- channels recorded from membrane patches of neonatal rat cortical neurons in culture. *J Pharmacol Exp Ther* **243**, 1195-1201.

Yung WH, Hausser MA and Jack JJB (1991). Electrophysiology of dopaminergic and non-dopaminergic neurons of the guinea-pig substantia nigra pars compacta *in vitro*. *J Physiol (Lond)* **436**, 643-667.

Appendix

This appendix includes the codes for the Automatic Detection software. The following paragraphs summarize how the algorithm is implemented. For the complete algorithm, please read the appended codes.

The data points is stored in the array:

```
REAL_DATA_TYPE real_data_array[DATA_SECTION_POINT_NO_LIMIT]
```

The *data window* is defined by the function:

```
int detect_data_window(REAL_DATA_TYPE
                       *pointer_real_data_window_array,
                        int array_limit_no,
                        double time_interval,
                        double window_time_width)
```

The base line current of the IPSCs is found by the function:

```
double averaged_base_line_current(REAL_DATA_TYPE
                                  *pointer_test_point,
                                   int array_limit)
```

The peak of the IPSCs is found by the function:

```
REAL_DATA_TYPE *pointer_fn_find_tempt_peak(REAL_DATA_TYPE
                                           *pointer_test_point,
                                           int array_limit)
```

The rising phase of the IPSCs is examined by the function:

```
int find_rising_phase(REAL_DATA_TYPE *pointer_star_point)
```

The decay phase of the IPSCs is examined by the function:

```
int find_falling_phase(REAL_DATA_TYPE *pointer_star_point)
```

Whether two events overlap together is tested by the function:

```
int test_overlapping_previouse_event(REAL_DATA_TYPE
                                     *pointer_test_point)
```

The 20-80% rise time is found by the function:

```
REAL_DATA_TYPE *pointer_fn_find_rise_time(REAL_DATA_TYPE
                                          *pointer_test_point)
```

The decay time constant is found by the function:

```
void find_decay_time_constant(REAL_DATA_TYPE *pointer_to_event)
```

```
// AutomaticDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Automatic.h"
#include "AutomaticDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


/////////////////////////////////////////////////
// MY CODE STARTS HERE
//////

/////////////////////////////////////////////////
////////////////included file
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

////////////////included file
/////////////////////////////////////////////////

#define DATA_SECTION_POINT_NO_LIMIT 11000
#define EVENT_NO_LIMIT 8100

int     repaint_value = 0;  //allow repaint trace if equal to 1
                            // if value equal to 0, do not allow

char    read_file_name[200];        //file name to open
FILE    *fpin;

/////////////////////////////////////////////////////////////
///// variables declaration for data section parameters
long    file_data_section_no;

long    GetDSChan_startOffset;
long    GetDSChan_points;
float   GetDSChan_yScale;
float   GetDSChan_yOffset;
float   GetDSChan_xScale;
float   GetDSChan_xOffset;


long    *p_file_data_section_no;

long    *file_GetDSChan_points;
float   *file_GetDSChan_yScale;
float   *file_GetDSChan_yOffset;
float   *file_GetDSChan_xScale;
float   *file_GetDSChan_xOffset;


char    file_yUnits[20];
char    file_xUnits[20];
short   file_data_point_spacing;

int     data_section_total_point_no;
/////// variables declaration for data section parameters
/////////////////////////////////////////////////////////////

size_t  byte_data_star = 64;
size_t  byte_to_start_read;
long    data_section_to_read;
long    temp_m_current_data_section_no;
long    foward_data_section_no;
long    back_data_section_no;

short   data_in_array[DATA_SECTION_POINT_NO_LIMIT];
long    nread;
int     seek_value;

/////////////////////////////////////////////////////////////
////////////display function variables//////////////////
int     display_point_array[DATA_SECTION_POINT_NO_LIMIT];
int     origin_x = 0;
int     origin_y = 120;
size_t  size_of_point;

int     x_position;

int     x_axis_scale_factor;
int     x_axis_percentage_constant = 100;
```

```
int     data_section_width_pixel_no = 500;

int     x_axis_display_star_factor;
int     x_axis_display_star_constant = 10000;
int     x_axis_display_star_point_no;
int     x_axis_total_display_point_no;

int     y_axis_scale_factor;
int     y_axis_base_factor = 20;

int     y_axis_percentage_constant = 100;
int     y_axis_position_offset;

int     increase_x_axis_scale;
int     decrease_x_axis_scale;



int     increase_y_axis_scale;
int     decrease_y_axis_scale;

int     increase_y_axis_position_offset;
int     decrease_y_axis_position_offset;

////////////display function variables//////////////////
//////////////////////////////////////////////////////////



//////////////////////////////////////////////////////
//////////////Define fitted data type
typedef struct {
    double  decay_time_constant;


} FITTED_DATA_TYPE;

//////////////Define fitted data type
//////////////////////////////////////////////////////




//////////////////////////////////////////////////////
//////////////Define Data structure
typedef struct {
    int     index;
    int     file_data_value;

    long    section_no;
    double  time;
    double  time_interval;
    double  time_local;

    double  current;
    double  averaged_current;

    double  slope;
    double  general_slope;

    int     screen_x_position;
    int     screen_y_position;


//////////////////////////////////////////////////////
////////////define event//////////////////////
    int     find_event_tempt_result;
    int     find_event_result;

    int     overlapping_previouse_event_result;
    int     overlapping_foward_event_result;

    void    *pointer_to_overlapping_previouse_event;
    void    *pointer_to_overlapping_foward_event;

    void    *pointer_to_event_backward_limit;
    void    *pointer_to_event_forward_limit;
    int     find_event_forward_limit_result;

    void    *pointer_to_belonged_event;
    int     test_belonged_to_event_result;
////////////define event//////////////////////
//////////////////////////////////////////////////////
```

```
    double  base_line_current;
    double  amplitude;

    void    *pointer_to_decay_time;
    double  decay_time;
    int     find_decay_time_result;

    void    *pointer_to_rise_time;
    void    *pointer_to_proximal_rise_time;
    int     find_rise_time_result;
    int     find_proximal_rise_time_result;
    double  rise_time;

    void    *pointer_to_begin_fit_decay;
    double  event_local_time;
    double  ideal_fitted_current;
    double  decay_time_constant;
    double  mean_decay_current_error_2;

    double  inter_event_interval;

} REAL_DATA_TYPE;

REAL_DATA_TYPE  real_data_array[DATA_SECTION_POINT_NO_LIMIT];
//////////////Define Data structure
////////////////////////////////////////////////////



////////////////////////////////////////////////////
//////////////event array
REAL_DATA_TYPE  event_array[EVENT_NO_LIMIT];
REAL_DATA_TYPE  *pointer_write_event;

long begin_detect_data_section_no;
long final_detected_data_section_no;
long no_of_data_section_for_detection;


long    written_event_no;
double  total_time_for_detection;
//////////////event array
////////////////////////////////////////////////////



////////////////////////////////////////////////////
//////////////total event
REAL_DATA_TYPE  total_event_array[EVENT_NO_LIMIT];
REAL_DATA_TYPE  *pointer_write_total_event;

long    total_event_no;
double  total_event_frequency;

//////////////total event
////////////////////////////////////////////////////



////////////////////////////////////////////////////
//////////////overlapping event
long    overlapping_event_no;
double  overlapping_event_proportion;

//////////////overlapping event
////////////////////////////////////////////////////



////////////////////////////////////////////////////
//////////////Parameters for detection
double  global_threshold_amplitude; /// unit nA
double  global_rising_threshold_slope;  /// unit nA/s
double  global_falling_threshold_slope; /// unit nA/s

int     global_rising_threshold_point_no;
double  global_falling_time_proportion;
//////////////Parameters for detection
////////////////////////////////////////////////////


////////////////////////////////////////////////////
//////////////condition for detection
double  global_decay_time;
//////////////condition for detection
```

```
//////////////////////////////////////////////////////

//////////////////////////////////////////////////////
//////////////output file
char    summary_file_name[200];
FILE    *fp_summary_file;

char    event_file_name[200];
FILE    *fp_event_file;

char    total_event_file_name[200];
FILE    *fp_total_event_file;
//////////////output file
//////////////////////////////////////////////////////


/////
// MY CODE ENDS HERE
//////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAutomaticDlg dialog

CAutomaticDlg::CAutomaticDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CAutomaticDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CAutomaticDlg)
    m_read_file_name = _T("");
    m_total_data_section_no = 0;
    m_current_data_section_no = 0;
    m_y_axis_scale_factor = 0;
    m_y_axis_position_offset = 0;
    m_x_axis_scale_factor = 0;
    m_x_axis_display_star_factor = 0;
    m_x_axis_bar_value = 0;
    m_x_axis_bar_unit = _T("");
    m_y_axis_bar_value = 0;
    m_y_axis_bar_unit = _T("");
    m_threshold_amplitude = 0.0;
    m_rising_threshold_slope = 0.0;
```

```
    m_falling_threshold_slope = 0.0;
    m_rising_threshold_point_no = 0;
    m_falling_time_proportion = 0.0;
    m_decay_time = 0.0;
    m_final_detected_data_section_no = 0;
    m_summary_file_name = _T("");
    m_event_file_name = _T("");
    m_total_event_file_name = _T("");
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CAutomaticDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAutomaticDlg)
    DDX_Text(pDX, IDC_FILE_NAME_EDIT, m_read_file_name);
    DDX_Text(pDX, IDC_TOTAL_DATA_SECTION_NO_EDIT, m_total_data_section_no);
    DDX_Text(pDX, IDC_CURRENT_DATA_SECTION_NO_EDIT, m_current_data_section_no);
    DDV_MinMaxLong(pDX, m_current_data_section_no, 1, 1000000);
    DDX_Text(pDX, IDC_Y_AXIS_SCALE_FACTOR_EDIT, m_y_axis_scale_factor);
    DDV_MinMaxInt(pDX, m_y_axis_scale_factor, 1, 10000);
    DDX_Text(pDX, IDC_Y_AXIS_POSITION_OFFSET_EDIT, m_y_axis_position_offset);
    DDV_MinMaxInt(pDX, m_y_axis_position_offset, -10000, 10000);
    DDX_Text(pDX, IDC_X_AXIS_SCALE_FACTOR_EDIT, m_x_axis_scale_factor);
    DDV_MinMaxInt(pDX, m_x_axis_scale_factor, 1, 1000000);
    DDX_Text(pDX, IDC_X_AXIS_DISPLAY_STAR_FACTOR_EDIT, m_x_axis_display_star_factor);
    DDV_MinMaxInt(pDX, m_x_axis_display_star_factor, 0, 9999);
    DDX_Text(pDX, IDC_X_AXIS_BAR_VALUE_EDIT, m_x_axis_bar_value);
    DDV_MinMaxLong(pDX, m_x_axis_bar_value, 0, 1000000000);
    DDX_Text(pDX, IDC_X_AXIS_BAR_UNIT_EDIT, m_x_axis_bar_unit);
    DDX_Text(pDX, IDC_Y_AXIS_BAR_VALUE_EDIT, m_y_axis_bar_value);
    DDV_MinMaxLong(pDX, m_y_axis_bar_value, 0, 1000000000);
    DDX_Text(pDX, IDC_Y_AXIS_BAR_UNIT_EDIT, m_y_axis_bar_unit);
    DDX_Text(pDX, IDC_THRESHOLD_AMPLITUDE_EDIT, m_threshold_amplitude);
    DDV_MinMaxDouble(pDX, m_threshold_amplitude, -1000000000., 1000000000.);
    DDX_Text(pDX, IDC_RISING_THRESHOLD_SLOPE_EDIT, m_rising_threshold_slope);
    DDV_MinMaxDouble(pDX, m_rising_threshold_slope, -1000000000., 1000000000.);
    DDX_Text(pDX, IDC_FALLING_THRESHOLD_SLOPE_EDIT, m_falling_threshold_slope);
    DDV_MinMaxDouble(pDX, m_falling_threshold_slope, -1000000000., 1000000000.);
    DDX_Text(pDX, IDC_RISING_THRESHOLD_POINT_NO_EDIT, m_rising_threshold_point_no);
    DDV_MinMaxInt(pDX, m_rising_threshold_point_no, 0, 10000);
    DDX_Text(pDX, IDC_FALLING_TIME_PROPORTION_EDIT, m_falling_time_proportion);
    DDV_MinMaxDouble(pDX, m_falling_time_proportion, 0., 1.);
    DDX_Text(pDX, IDC_DECAY_TIME_EDIT, m_decay_time);
    DDV_MinMaxDouble(pDX, m_decay_time, 1.e-002, 0.99);
    DDX_Text(pDX, IDC_FINAL_DETECTED_DATA_SECTION_NO_EDIT, m_final_detected_data_secti
on_no);
    DDV_MinMaxLong(pDX, m_final_detected_data_section_no, 1, 1000000);
    DDX_Text(pDX, IDC_SUMMARY_FILE_NAME_EDIT, m_summary_file_name);
    DDX_Text(pDX, IDC_EVENT_FILE_NAME_EDIT, m_event_file_name);
    DDX_Text(pDX, IDC_TOTAL_EVENT_FILE_NAME_EDIT, m_total_event_file_name);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAutomaticDlg, CDialog)
    //{{AFX_MSG_MAP(CAutomaticDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_DISPLAY_BUTTON, OnDisplayButton)
    ON_EN_CHANGE(IDC_FILE_NAME_EDIT, OnChangeFileNameEdit)
    ON_BN_CLICKED(IDC_OPEN_BUTTON, OnOpenButton)
    ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
    ON_EN_CHANGE(IDC_CURRENT_DATA_SECTION_NO_EDIT, OnChangeCurrentDataSectionNoEdit)
    ON_EN_KILLFOCUS(IDC_CURRENT_DATA_SECTION_NO_EDIT, OnKillfocusCurrentDataSectionNoE
dit)
    ON_BN_CLICKED(IDC_UPDATE_SCREEN_DATA_SECTION_BUTTON, OnUpdateScreenDataSectionButt
on)
    ON_BN_CLICKED(IDC_DATA_SECTION_FORWARD_BUTTON, OnDataSectionForwardButton)
    ON_BN_CLICKED(IDC_DATA_SECTION_BACK_BUTTON, OnDataSectionBackButton)
    ON_EN_KILLFOCUS(IDC_Y_AXIS_SCALE_FACTOR_EDIT, OnKillfocusYAxisScaleFactorEdit)
    ON_EN_KILLFOCUS(IDC_Y_AXIS_POSITION_OFFSET_EDIT, OnKillfocusYAxisPositionOffsetEdi
t)
    ON_EN_KILLFOCUS(IDC_X_AXIS_SCALE_FACTOR_EDIT, OnKillfocusXAxisScaleFactorEdit)
    ON_EN_KILLFOCUS(IDC_X_AXIS_DISPLAY_STAR_FACTOR_EDIT, OnKillfocusXAxisDisplayStarFa
ctorEdit)
    ON_BN_CLICKED(IDC_INCREASE_Y_AXIS_SCALE_BUTTON, OnIncreaseYAxisScaleButton)
    ON_BN_CLICKED(IDC_DECREASE_Y_AXIS_SCALE_BUTTON, OnDecreaseYAxisScaleButton)
    ON_BN_CLICKED(IDC_INCREASE_Y_AXIS_POSITION_BUTTON, OnIncreaseYAxisPositionButton)
    ON_BN_CLICKED(IDC_DECREASE_Y_AXIS_POSITION_BUTTON, OnDecreaseYAxisPositionButton)
    ON_BN_CLICKED(IDC_INCREASE_X_AXIS_SCALE_BUTTON, OnIncreaseXAxisScaleButton)
    ON_BN_CLICKED(IDC_DECREASE_X_AXIS_SCALE_BUTTON, OnDecreaseXAxisScaleButton)
    ON_BN_CLICKED(IDC_INCREASE_X_AXIS_STAR_BUTTON, OnIncreaseXAxisStarButton)
    ON_BN_CLICKED(IDC_DECREASE_X_AXIS_STAR_BUTTON, OnDecreaseXAxisStarButton)
    ON_BN_CLICKED(IDC_DETECT_BUTTON, OnDetectButton)
    ON_EN_KILLFOCUS(IDC_THRESHOLD_AMPLITUDE_EDIT, OnKillfocusThresholdAmplitudeEdit)
```

```
    ON_EN_KILLFOCUS(IDC_RISING_THRESHOLD_SLOPE_EDIT, OnKillfocusRisingThresholdSlopeEd
it)
    ON_EN_KILLFOCUS(IDC_FALLING_THRESHOLD_SLOPE_EDIT, OnKillfocusFallingThresholdSlope
Edit)
    ON_EN_KILLFOCUS(IDC_RISING_THRESHOLD_POINT_NO_EDIT, OnKillfocusRisingThresholdPoin
tNoEdit)
    ON_EN_KILLFOCUS(IDC_FALLING_TIME_PROPORTION_EDIT, OnKillfocusFallingTimeProportion
Edit)
    ON_EN_KILLFOCUS(IDC_DECAY_TIME_EDIT, OnKillfocusDecayTimeEdit)
    ON_EN_KILLFOCUS(IDC_FINAL_DETECTED_DATA_SECTION_NO_EDIT, OnKillfocusFinalDetectedD
ataSectionNoEdit)
    ON_BN_CLICKED(IDC_DETECT_SECTIONS_BUTTON, OnDetectSectionsButton)
    ON_EN_CHANGE(IDC_SUMMARY_FILE_NAME_EDIT, OnChangeSummaryFileNameEdit)
    ON_EN_CHANGE(IDC_EVENT_FILE_NAME_EDIT, OnChangeEventFileNameEdit)
    ON_BN_CLICKED(IDC_SAVE_OUTPUT_FILE_BUTTON, OnSaveOutputFileButton)
    ON_EN_CHANGE(IDC_TOTAL_EVENT_FILE_NAME_EDIT, OnChangeTotalEventFileNameEdit)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAutomaticDlg message handlers

BOOL CAutomaticDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);         // Set big icon
    SetIcon(m_hIcon, FALSE);        // Set small icon

    // TODO: Add extra initialization here

////////////////////////////////////////////
// MY CODE STARTS HERE
//////

/////////////////////////////////////////////////
///////////////initialise the control variables
    m_current_data_section_no = 1;

    m_x_axis_scale_factor = 100;
    x_axis_scale_factor = m_x_axis_scale_factor;

    m_x_axis_display_star_factor = 0;
    x_axis_display_star_factor = m_x_axis_display_star_factor;

    m_y_axis_scale_factor = 100;
    y_axis_scale_factor = m_y_axis_scale_factor;

    m_y_axis_position_offset = 0;
    y_axis_position_offset = m_y_axis_position_offset;

///////////////initialise the control variables
/////////////////////////////////////////////////

/////////////////////////////////////////////////
///////initialise the control variables for detection

    m_threshold_amplitude = 0.03;   /// unit nA
    global_threshold_amplitude = m_threshold_amplitude;

    m_rising_threshold_slope = -10; /// unit nA/s
    global_rising_threshold_slope = m_rising_threshold_slope;

    m_falling_threshold_slope = 0.5;    /// unit nA/s
    global_falling_threshold_slope = m_falling_threshold_slope;


    m_rising_threshold_point_no = 2;
```

```
    global_rising_threshold_point_no = m_rising_threshold_point_no;

    m_falling_time_proportion = 0.5;
    global_falling_time_proportion = m_falling_time_proportion;

///////initialise the control variables for detection
////////////////////////////////////////////////


////////////////////////////////////////////////
//////////////condition for detection

    m_decay_time = 0.5;
    global_decay_time = m_decay_time;

    m_final_detected_data_section_no = 1;

//////////////condition for detection
////////////////////////////////////////////////

////////////////////////////////////////////////
//////////////output file

    m_summary_file_name = "summary.txt";
    m_event_file_name = "event.txt";
    m_total_event_file_name = "total.txt";

//////////////output file
////////////////////////////////////////////////

    UpdateData(FALSE);      // Update the screen

/////
// MY CODE ENDS HERE
////////////////////////////////////////////////


    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CAutomaticDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CAutomaticDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
////////////////////////////////////////////
// MY CODE STARTS HERE
//////

////////////////////////////////////////////////
////to call the display function after opening the file
        if (repaint_value == 1)
            {
                CAutomaticDlg::OnDisplayButton();
```

```
            }
////to call the display function after opening the file
//////////////////////////////////////////////////

/////
// MY CODE ENDS HERE
////////////////////////////////////////////////
        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CAutomaticDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CAutomaticDlg::OnChangeFileNameEdit()
{
    // TODO: If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialog::OnInitDialog()
    // function to send the EM_SETEVENTMASK message to the control
    // with the ENM_CHANGE flag ORed into the lParam mask.

    // TODO: Add your control notification handler code here

////////////////////////////////////////////
// MY CODE STARTS HERE
//////

    // Update the variables of the controls.
    UpdateData(TRUE);

/////
// MY CODE ENDS HERE
////////////////////////////////////////////////
}

void CAutomaticDlg::OnOpenButton()
{
    // TODO: Add your control notification handler code here

////////////////////////////////////////////
// MY CODE STARTS HERE
//////
    int counter_initial_real_data_array;

    for(counter_initial_real_data_array = 0;
        counter_initial_real_data_array <= (DATA_SECTION_POINT_NO_LIMIT - 1);
        counter_initial_real_data_array++)
        {
            real_data_array[counter_initial_real_data_array].file_data_value = 0;

        }

    p_file_data_section_no = &file_data_section_no;

    file_GetDSChan_points = &GetDSChan_points;
    file_GetDSChan_yScale = &GetDSChan_yScale;
    file_GetDSChan_yOffset = &GetDSChan_yOffset;
    file_GetDSChan_xScale = &GetDSChan_xScale;
    file_GetDSChan_xOffset = &GetDSChan_xOffset;

    strcpy(read_file_name, (LPCTSTR)m_read_file_name);
    fpin = fopen(read_file_name, "rb");
        if(fpin == NULL)
            {
                MessageBox(" Error in opening file, check the file name! ");
                return;
            }

        fread(p_file_data_section_no, sizeof(long), 1, fpin);

        fread(file_GetDSChan_points, sizeof(long), 1, fpin);
        fread(file_GetDSChan_yScale, sizeof(float), 1, fpin);
        fread(file_GetDSChan_yOffset, sizeof(float), 1, fpin);
        fread(file_GetDSChan_xScale, sizeof(float), 1, fpin);
        fread(file_GetDSChan_xOffset, sizeof(float), 1, fpin);

        fread(file_yUnits, sizeof(char), 20, fpin);
        fread(file_xUnits, sizeof(char), 20, fpin);

    fclose(fpin);

    data_section_total_point_no = (int)GetDSChan_points;
    repaint_value = 1;
```

```
//////////////////////////////////////////////////////////////////////
///////////////to initialise the data section//////////////////////////
    m_total_data_section_no = file_data_section_no;
    GetDlgItem(IDC_TOTAL_DATA_SECTION_NO_EDIT)->EnableWindow(FALSE);

    m_current_data_section_no = 1;
    data_section_to_read = m_current_data_section_no;

    m_x_axis_bar_unit = (CString)"1E-6" + file_xUnits;
    GetDlgItem(IDC_X_AXIS_BAR_UNIT_EDIT)->EnableWindow(FALSE);

    GetDlgItem(IDC_X_AXIS_BAR_VALUE_EDIT)->EnableWindow(FALSE);


    m_y_axis_bar_unit = (CString)"1E-6" + file_yUnits;
    GetDlgItem(IDC_Y_AXIS_BAR_UNIT_EDIT)->EnableWindow(FALSE);

    GetDlgItem(IDC_Y_AXIS_BAR_VALUE_EDIT)->EnableWindow(FALSE);


    UpdateData(FALSE);      // Update the screen
///////////////to initialise the data section//////////////////////////
//////////////////////////////////////////////////////////////////////

    CAutomaticDlg::read_data(data_section_to_read);
    Invalidate();   // Cause the execution of the OnPaint() function

/////
// MY CODE ENDS HERE
//////////////////////////////////////////////////////////////
}


//////////////////////////////////////////////////////////
// MY CODE STARTS HERE
////////
///a function to convert the file data value to the
///real current with the Unit defined inside the file
double convert_file_data_value_to_real_current(int file_data_value)
{
double  real_current;

        real_current = (double)((double)file_data_value *
            (double)GetDSChan_yScale + (double)GetDSChan_yOffset);

        return (real_current);
}
///a function to convert the file data value to the
///real current with the Unit defined inside the file
/////
// MY CODE ENDS HERE
//////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
////////
///a function to convert the point number within the data section
///to the real time with unit defined within the data file
double convert_data_section_point_no_to_real_time(int data_section_point_no)
{

double  real_time;

        real_time = ((double)(data_section_point_no + 1) * (double)GetDSChan_xScale);

        return (real_time);
}
///a function to convert the point number within the data section
///to the real time with unit defined within the data file
/////
// MY CODE ENDS HERE
//////////////////////////////////////////////////////////////



//////////////////////////////////////////
// MY CODE STARTS HERE
//////////////////////////read data function ////////////////////
void CAutomaticDlg::read_data(long fn_read_data_data_section_no)
{
double slope_of_the_point(REAL_DATA_TYPE *, double);
double general_slope_of_the_point(REAL_DATA_TYPE *, double);

double convert_data_section_point_no_to_real_time(int);
double convert_file_data_value_to_real_current(int);

    byte_to_start_read = (byte_data_star +
```

```
                        (((size_t)(fn_read_data_data_section_no - 1)) *
                        ((size_t)GetDSChan_points * sizeof(short))));

    fpin = fopen(read_file_name, "rb");
        if(fpin == NULL)
            {
                 MessageBox(" Error in opening file! ");
                 return;
            }

        seek_value = fseek(fpin, byte_to_start_read, SEEK_SET);

        if(seek_value != 0)
            {
            MessageBox(" Error in fseek! ");
            }

        nread = fread(data_in_array, sizeof(short), GetDSChan_points, fpin);

    fclose(fpin);

    if(nread != GetDSChan_points)
        {
         MessageBox(" Error in fread! ");
        }



    int counter_intialise_element;

    for(counter_intialise_element = 0;
        counter_intialise_element <= (data_section_total_point_no - 1);
        counter_intialise_element++)
        {
            display_point_array[counter_intialise_element] =
                            (int)data_in_array[counter_intialise_element];

            real_data_array[counter_intialise_element].index =
                                counter_intialise_element;

            real_data_array[counter_intialise_element].file_data_value =
                            (int)data_in_array[counter_intialise_element];

            real_data_array[counter_intialise_element].section_no =
                            fn_read_data_data_section_no;

            real_data_array[counter_intialise_element].time =
                convert_data_section_point_no_to_real_time(
                                        counter_intialise_element);

            real_data_array[counter_intialise_element].current =
                    convert_file_data_value_to_real_current(
                    real_data_array[counter_intialise_element].file_data_value);

            real_data_array[counter_intialise_element].time_interval =
                            (double)GetDSChan_xScale;
        }


void initialising_detection_paramers();

    initialising_detection_paramers();


    for(counter_intialise_element = 0;
        counter_intialise_element <= (data_section_total_point_no - 2);
        counter_intialise_element++)
        {

            real_data_array[counter_intialise_element].slope =
                slope_of_the_point(&real_data_array[counter_intialise_element],
                       ((double)GetDSChan_xScale));
        }

    for(counter_intialise_element = 6;
        counter_intialise_element <= (data_section_total_point_no - 6);
        counter_intialise_element++)
        {

            real_data_array[counter_intialise_element].general_slope =
                general_slope_of_the_point(&real_data_array[counter_intialise_element]
,
                                    ((double)GetDSChan_xScale));
        }

double averaging_current(REAL_DATA_TYPE *);

    for(counter_intialise_element = 1;
```

```
        counter_intialise_element <= (data_section_total_point_no - 2);
        counter_intialise_element++)
        {

            real_data_array[counter_intialise_element].averaged_current =
                averaging_current(&real_data_array[counter_intialise_element]);

        }

}
//////////////////////////read data function /////////////////////
/////
// MY CODE ENDS HERE
////////////////////////////////////////////////////////////







////////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
////////
///a function to convert the file data value to the
///screen position
int convert_file_data_value_to_screen_position(int file_data_value)
{
int     y_position;
int     returned_screen_position;

        y_position = (file_data_value *
                     y_axis_scale_factor)
                     /
                     (y_axis_base_factor * y_axis_percentage_constant);

        returned_screen_position = -y_position + origin_y - y_axis_position_offset;
        return (returned_screen_position);
}
///a function to convert the file data value to the
///screen position
/////
// MY CODE ENDS HERE
////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to set the screen positions of the data points
void set_data_screen_position(REAL_DATA_TYPE *pointer_real_data_array,
                              int array_limit_no)
{
int convert_file_data_value_to_screen_position(int);

int temp_screen_x_position[DATA_SECTION_POINT_NO_LIMIT];
int x_offset;

    int counter_set_zero;

    for(counter_set_zero = 0;
        counter_set_zero <= (array_limit_no - 1);
        counter_set_zero++)
        {
            (pointer_real_data_array + counter_set_zero)->screen_x_position = 0;
            (pointer_real_data_array + counter_set_zero)->screen_y_position = 0;
        }


    int counter_position;

    for(counter_position = 0;
        counter_position <= (array_limit_no - 1);
        counter_position++)
        {
            temp_screen_x_position[counter_position] =
                    (int)(((double)(counter_position + 1)*
                     (double)data_section_width_pixel_no *
                     (double)x_axis_scale_factor)
                     /
                     ((double)x_axis_percentage_constant *
                     (double)array_limit_no));
        }

    x_axis_display_star_point_no =
                    ((x_axis_display_star_factor * data_section_total_point_no)
```

```
                    / x_axis_display_star_constant);

    x_offset = temp_screen_x_position[x_axis_display_star_point_no];

    for(counter_position = 0;
        counter_position <= (array_limit_no - 1);
        counter_position++)
        {
            (pointer_real_data_array + counter_position)->screen_x_position =
                temp_screen_x_position[counter_position] - x_offset + 1 + origin_x;

            if ((pointer_real_data_array + counter_position)->screen_x_position < 0)
                {
                    (pointer_real_data_array + counter_position)->screen_x_position =
0;
                }

            if ((pointer_real_data_array + counter_position)->screen_x_position >
                                                    data_section_width_pixel_no)
                {
                    (pointer_real_data_array + counter_position)->screen_x_position =
                                    data_section_width_pixel_no;
                }
        }


    for(counter_position = 0;
        counter_position <= (array_limit_no - 1);
        counter_position++)
        {
            (pointer_real_data_array + counter_position)->screen_y_position =
                convert_file_data_value_to_screen_position((pointer_real_data_array +
                counter_position)->file_data_value);
        }

}
///a function to set the screen positions of the data points
// MY CODE ENDS HERE
////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to convert the current  to the
///screen y position
int current_to_screen_y_position(double current)
{
int     convert_file_data_value_to_screen_position(int);

double  data_value;
int     screen_y_position;

        data_value = (current - (double)GetDSChan_yOffset)
                        /(double)GetDSChan_yScale;

        screen_y_position =
            convert_file_data_value_to_screen_position((int)data_value);
        return (screen_y_position);
}
///a function to convert the current  to the
///screen y position
// MY CODE ENDS HERE
////////////////////////////////////////////////////////////////



void CAutomaticDlg::OnDisplayButton()
{
    // TODO: Add your control notification handler code here

////////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
////////
int convert_file_data_value_to_screen_position(int);
void set_data_screen_position(REAL_DATA_TYPE *, int);
int current_to_screen_y_position(double);

//  CClientDC   dc(this);   // Create a dc object
    CPaintDC dc(this); // device context for painting

    set_data_screen_position(real_data_array, data_section_total_point_no);

        m_star_x = real_data_array[0].screen_x_position;
        m_star_y = real_data_array[0].screen_y_position;

    int counter_display_loop;
```

```
    for(counter_display_loop = 0;
        counter_display_loop <= (data_section_total_point_no - 1);
        counter_display_loop++)
        {
            m_end_x = real_data_array[counter_display_loop].screen_x_position;
            m_end_y = real_data_array[counter_display_loop].screen_y_position;

            dc.MoveTo(m_star_x, m_star_y);
            dc.LineTo(m_end_x, m_end_y);

            m_star_x = m_end_x;
            m_star_y = m_end_y;
        }

////////////////////////////////////////////////////////////////
//////////////////////////draw the zero line//////////////////////
int     data_vaule_corresponding_to_0_current;
int     screen_position_corresponding_to_0_current;

    data_vaule_corresponding_to_0_current =
    (int)(-GetDSChan_yOffset / GetDSChan_yScale);

    screen_position_corresponding_to_0_current =
    convert_file_data_value_to_screen_position(data_vaule_corresponding_to_0_current);

    dc.MoveTo((origin_x + 600), screen_position_corresponding_to_0_current);
    dc.LineTo((origin_x + 600 + 50), screen_position_corresponding_to_0_current);
//////////////////////////draw the zero line//////////////////////
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
//////////////////////////draw X Axis scale bar//////////////////////
int     x_axis_bar_length = 50;

    dc.MoveTo((origin_x + 550), (origin_y +70));
    dc.LineTo((origin_x + 550 + x_axis_bar_length), (origin_y +70));

    m_x_axis_bar_value =
        (long)((double)GetDSChan_xScale *
            (double)data_section_total_point_no *
            ((double)x_axis_bar_length / (double)data_section_width_pixel_no) *
            ((double)x_axis_percentage_constant /(double)m_x_axis_scale_factor) *
            1000000);

    UpdateData(FALSE);      // Update the screen
//////////////////////////draw X Axis scale bar//////////////////////
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
//////////////////////////draw Y Axis scale bar//////////////////////
double  y_axis_bar_length_double;
int     y_axis_bar_length_int;

    /////////set the y_axis_bar_length so that it
    /////////represent 0.2 nA
    y_axis_bar_length_double =
        (double)((double)0.2 * (double)1
            /
            ((double)y_axis_base_factor * (double)GetDSChan_yScale));

    m_y_axis_bar_value =
        (long)((double)y_axis_bar_length_double  *
            (double)y_axis_base_factor *
            ((double)y_axis_percentage_constant / (double)y_axis_scale_factor) *
            (double)GetDSChan_yScale *
            1000000);

    UpdateData(FALSE);      // Update the screen

    y_axis_bar_length_int = (int)y_axis_bar_length_double;

    dc.MoveTo((origin_x + 550), (origin_y +70));
    dc.LineTo((origin_x + 550), (origin_y +70 - y_axis_bar_length_int));
//////////////////////////draw Y Axis scale bar//////////////////////
////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////
//////////////////////////display the events//////////////////////

    int event_x_position;
    int event_y_position;


    // Create a new pen
    CPen MyNewPen;
```

```
    MyNewPen.CreatePen (PS_SOLID,
                        2,
                        RGB(255,0,0) );

    // Select the new pen.
    CPen* pOriginalPen;
    pOriginalPen = dc.SelectObject(&MyNewPen);


    int counter;

    for(counter = 0;
        counter <= (data_section_total_point_no - 1);
        counter++)
        {
            if (
                (real_data_array[counter].find_event_result == 1)
                &&
                (real_data_array[counter].overlapping_previouse_event_result == 0)
                &&
                (real_data_array[counter].overlapping_foward_event_result == 0)
                )
                {
                    event_x_position = real_data_array[counter].screen_x_position;
                    event_y_position =
                            current_to_screen_y_position(
                            real_data_array[counter].averaged_current);


                    CRect MyRctangle (event_x_position - 2,
                                      event_y_position - 2,
                                      event_x_position + 2,
                                      event_y_position + 2);

                    // Draw the circle
                    dc.Ellipse (&MyRctangle);

                }
        }

    // Return the original pen
    dc.SelectObject(&pOriginalPen);

//////////////////////////display the events////////////////////
////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////
//////////////////////////display the ovelapping events////////////////////

    int ovelapping_event_x_position;
    int ovelapping_event_y_position;



    // Create a new pen
    CPen MyNewPen_ovelapping_event;

    MyNewPen_ovelapping_event.CreatePen (PS_SOLID,
                                         2,
                                         RGB(255,150,0) );

    // Select the new pen.
    CPen* pOriginalPen_ovelapping_event;
    pOriginalPen_ovelapping_event = dc.SelectObject(&MyNewPen_ovelapping_event);


    for(counter = 0;
        counter <= (data_section_total_point_no - 1);
        counter++)
        {
            if (
                (real_data_array[counter].find_event_result == 1)
                        &&
                    (
```

```
                    (real_data_array[counter].overlapping_previouse_event_result == 1)
                    ||
                    (real_data_array[counter].overlapping_foward_event_result == 1)
                    )
                )
                {
                    ovelapping_event_x_position = real_data_array[counter].screen_x_po
sition;
                    ovelapping_event_y_position =
                            current_to_screen_y_position(
                            real_data_array[counter].averaged_current);


                    CRect MyRctangle_ovelapping_event (ovelapping_event_x_position - 2
,
                                                       ovelapping_event_y_position -
2,
                                                       ovelapping_event_x_position +
2,
                                                       ovelapping_event_y_position +
2);

                    // Draw the circle
                    dc.Ellipse (&MyRctangle_ovelapping_event);

                }
        }

    // Return the original pen
    dc.SelectObject(&pOriginalPen_ovelapping_event);

//////////////////////////display the ovelapping events////////////////////
////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////
//////////////////////////display the base line current/////////

    int base_line_x_position;
    int base_line_y_position;



    // Create a new pen
    CPen MyNewPen_base_line;

    MyNewPen_base_line.CreatePen (PS_SOLID,
                                  2,
                                  RGB(255,0,0) );

    // Select the new pen.
    CPen* pOriginalPen_base_line;
    pOriginalPen_base_line = dc.SelectObject(&MyNewPen_base_line);


    for(counter = 0;
        counter <= (data_section_total_point_no - 1);
        counter++)
        {
            if (real_data_array[counter].find_event_result == 1)
                {
                    base_line_x_position = real_data_array[counter].screen_x_position;

                    base_line_y_position =
                            current_to_screen_y_position(
                            real_data_array[counter].base_line_current);


                    CRect MyRctangle_base_line (base_line_x_position - 1,
                                                base_line_y_position - 1,
                                                base_line_x_position + 1,
                                                base_line_y_position + 1);

                    // Draw the circle
                    dc.Ellipse (&MyRctangle_base_line);

                }
        }

    // Return the original pen
    dc.SelectObject(&pOriginalPen_base_line);

//////////////////////////display the base line current/////////
////////////////////////////////////////////////////////////////////
```

```
/////////////////////////////////////////////////////////////////
///////////////////////////display the decay fittting curve//////////////


    // Create a new pen
    CPen MyNewPen_fitting_decay;

    MyNewPen_fitting_decay.CreatePen (PS_SOLID,
                                      1,
                                      RGB(255,255,0) );

    // Select the new pen.
    CPen* pOriginalPen_fitting_decay;
    pOriginalPen_fitting_decay = dc.SelectObject(&MyNewPen_fitting_decay);


    for(counter = 0;
        counter <= (data_section_total_point_no - 1);
        counter++)
        {///begin first for loop statement
            if (
                (real_data_array[counter].find_event_result == 1)
                &&
                (real_data_array[counter].overlapping_previouse_event_result == 0)
                &&
                (real_data_array[counter].overlapping_foward_event_result == 0)
                &&
                (real_data_array[counter].find_event_forward_limit_result == 1)
                )
                {///begin if loop statement


                    m_star_x = (  (REAL_DATA_TYPE *)(real_data_array[counter].
                                pointer_to_begin_fit_decay)  )->screen_x_position;

                    m_star_y = current_to_screen_y_position(
                            (  (REAL_DATA_TYPE *)(real_data_array[counter].
                            pointer_to_begin_fit_decay)  )->ideal_fitted_current
                            );



                    REAL_DATA_TYPE *pointer_counter;


                    for(pointer_counter = (  (REAL_DATA_TYPE *)(real_data_array[counte
r].
                                        pointer_to_begin_fit_decay)  );
                        pointer_counter <= (  (REAL_DATA_TYPE *)(real_data_array[count
er].
                                        pointer_to_event_forward_limit)  );
                        pointer_counter++)
                        {
                            m_end_x = pointer_counter->screen_x_position;
                            m_end_y = current_to_screen_y_position(
                                        pointer_counter->ideal_fitted_current);


                            dc.MoveTo(m_star_x, m_star_y);
                            dc.LineTo(m_end_x, m_end_y);

                            m_star_x = m_end_x;
                            m_star_y = m_end_y;
                        }


                }///end if loop statement
        }///end first for loop statement

    // Return the original pen
    dc.SelectObject(&pOriginalPen_fitting_decay);

///////////////////////////display the decay fittting curve//////////////
/////////////////////////////////////////////////////////////////







/////////////////////////////////////////////////////////////////
///////////////////////////display the decay time////////////////////
```

```
    int decay_time_x_position;
    int decay_time_y_position;



    // Create a new pen
    CPen MyNewPen_decay_time;

    MyNewPen_decay_time.CreatePen (PS_SOLID,
                                   2,
                                   RGB(0,255,0) );

    // Select the new pen.
    CPen* pOriginalPen_decay_time;
    pOriginalPen_decay_time = dc.SelectObject(&MyNewPen_decay_time);


    for(counter = 0;
        counter <= (data_section_total_point_no - 1);
        counter++)
        {
            if (real_data_array[counter].find_event_result == 1)
                {
                    decay_time_x_position =
                    ((REAL_DATA_TYPE *)(real_data_array[counter].pointer_to_decay_time
))
                    ->screen_x_position;

                    decay_time_y_position =
                            current_to_screen_y_position(
                            (  (REAL_DATA_TYPE *)(real_data_array[counter].
                                pointer_to_decay_time)  )->averaged_current);


                    CRect MyRctangle_decay_time (decay_time_x_position - 1,
                                                 decay_time_y_position - 1,
                                                 decay_time_x_position + 1,
                                                 decay_time_y_position + 1);

                    // Draw the circle
                    dc.Ellipse (&MyRctangle_decay_time);

                }
        }

    // Return the original pen
    dc.SelectObject(&pOriginalPen_decay_time);

//////////////////////////display the decay time//////////////////////
////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////
//////////////////////////display the rise time//////////////////////

    int rise_time_x_position;
    int rise_time_y_position;



    // Create a new pen
    CPen MyNewPen_rise_time;

    MyNewPen_rise_time.CreatePen (PS_SOLID,
                                  2,
                                  RGB(0,255,255) );

    // Select the new pen.
    CPen* pOriginalPen_rise_time;
    pOriginalPen_rise_time = dc.SelectObject(&MyNewPen_rise_time);


    for(counter = 0;
        counter <= (data_section_total_point_no - 1);
        counter++)
        {
            if (real_data_array[counter].find_event_result == 1)
                {
                    rise_time_x_position =
                    ((REAL_DATA_TYPE *)(real_data_array[counter].pointer_to_rise_time)
)
                    ->screen_x_position;

                    rise_time_y_position =
                    ((REAL_DATA_TYPE *)(real_data_array[counter].pointer_to_rise_time)
)
                    ->screen_y_position;
```

```
                    CRect MyRctangle_rise_time (rise_time_x_position - 1,
                                                rise_time_y_position - 1,
                                                rise_time_x_position + 1,
                                                rise_time_y_position + 1);

                    // Draw the circle
                    dc.Ellipse (&MyRctangle_rise_time);

                }
        }

    // Return the original pen
    dc.SelectObject(&pOriginalPen_rise_time);

////////////////////////////display the rise time//////////////////////
////////////////////////////////////////////////////////////////////////




////////////////////////////////////////////////////////////////////////
/////////////////display the proximal rise time//////////////////////

    int proximal_rise_time_x_position;
    int proximal_rise_time_y_position;



    // Create a new pen
    CPen MyNewPen_proximal_rise_time;

    MyNewPen_proximal_rise_time.CreatePen (PS_SOLID,
                                           2,
                                           RGB(255,0,255) );

    // Select the new pen.
    CPen* pOriginalPen_proximal_rise_time;
    pOriginalPen_proximal_rise_time = dc.SelectObject(&MyNewPen_proximal_rise_time);


    for(counter = 0;
        counter <= (data_section_total_point_no - 1);
        counter++)
        {
            if (real_data_array[counter].find_event_result == 1)
                {
                    proximal_rise_time_x_position =
                    ((REAL_DATA_TYPE *)(real_data_array[counter].pointer_to_proximal_r
ise_time))
                    ->screen_x_position;

                    proximal_rise_time_y_position =
                    ((REAL_DATA_TYPE *)(real_data_array[counter].pointer_to_proximal_r
ise_time))
                    ->screen_y_position;


                    CRect MyRctangle_proximal_rise_time (proximal_rise_time_x_position
 - 1,
                                                         proximal_rise_time_y_position - 1,
                                                         proximal_rise_time_x_position + 1,
                                                         proximal_rise_time_y_position + 1);

                    // Draw the circle
                    dc.Ellipse (&MyRctangle_proximal_rise_time);

                }
        }

    // Return the original pen
    dc.SelectObject(&pOriginalPen_proximal_rise_time);

/////////////////display the proximal rise time//////////////////////
////////////////////////////////////////////////////////////////////////


//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}
```

```
void CAutomaticDlg::OnExitButton()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////

    OnOK();        // Terminate the program


//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnChangeCurrentDataSectionNoEdit()
{
    // TODO: If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialog::OnInitDialog()
    // function to send the EM_SETEVENTMASK message to the control
    // with the ENM_CHANGE flag ORed into the lParam mask.

    // TODO: Add your control notification handler code here

}

void CAutomaticDlg::OnKillfocusCurrentDataSectionNoEdit()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////

    temp_m_current_data_section_no = m_current_data_section_no;
    UpdateData(TRUE);   // Update the variables of the controls.

    if (m_current_data_section_no > file_data_section_no)
        {
            m_current_data_section_no = temp_m_current_data_section_no;
            UpdateData(FALSE);       // Update the screen
        }
    else
        {
            data_section_to_read = m_current_data_section_no;
            CAutomaticDlg::read_data(data_section_to_read);
        }

    Invalidate();   // Cause the execution of the OnPaint() function
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////

}

void CAutomaticDlg::OnUpdateScreenDataSectionButton()
{
    // TODO: Add your control notification handler code here
//////////////////////////////////////////
// MY CODE STARTS HERE
////////

    Invalidate();   // Cause the execution of the OnPaint() function


//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnDataSectionForwardButton()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////

    foward_data_section_no = m_current_data_section_no + 1;

    if (foward_data_section_no <= file_data_section_no)
        {
            m_current_data_section_no = m_current_data_section_no + 1;
            UpdateData(FALSE);       // Update the screen
            data_section_to_read = m_current_data_section_no;
            CAutomaticDlg::read_data(data_section_to_read);
```

```
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}

void CAutomaticDlg::OnDataSectionBackButton()
{
    // TODO: Add your control notification handler code here

/////////////////////////////////////////
// MY CODE STARTS HERE
////////

    back_data_section_no = m_current_data_section_no - 1;

    if (back_data_section_no >= 1)
        {
            m_current_data_section_no = m_current_data_section_no - 1;
            UpdateData(FALSE);      // Update the screen
            data_section_to_read = m_current_data_section_no;
            CAutomaticDlg::read_data(data_section_to_read);
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}

void CAutomaticDlg::OnKillfocusYAxisScaleFactorEdit()
{
    // TODO: Add your control notification handler code here

/////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    y_axis_scale_factor = m_y_axis_scale_factor;
    Invalidate();   // Cause the execution of the OnPaint() function
//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}

void CAutomaticDlg::OnKillfocusYAxisPositionOffsetEdit()
{
    // TODO: Add your control notification handler code here

/////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    y_axis_position_offset = m_y_axis_position_offset;
    Invalidate();   // Cause the execution of the OnPaint() function
//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}

void CAutomaticDlg::OnKillfocusXAxisScaleFactorEdit()
{
    // TODO: Add your control notification handler code here

/////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    x_axis_scale_factor = m_x_axis_scale_factor;
    Invalidate();   // Cause the execution of the OnPaint() function
//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}


void CAutomaticDlg::OnKillfocusXAxisDisplayStarFactorEdit()
{
    // TODO: Add your control notification handler code here

/////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    x_axis_display_star_factor = m_x_axis_display_star_factor;
    Invalidate();   // Cause the execution of the OnPaint() function
//////////
// MY CODE ENDS HERE
```

```
//////////////////////////////////////////

}




void CAutomaticDlg::OnIncreaseYAxisScaleButton()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////

    increase_y_axis_scale = m_y_axis_scale_factor * 2;

    if (increase_y_axis_scale <= 10000)
        {
            m_y_axis_scale_factor = m_y_axis_scale_factor * 2;
            UpdateData(FALSE);      // Update the screen
            y_axis_scale_factor = m_y_axis_scale_factor;
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnDecreaseYAxisScaleButton()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////

    decrease_y_axis_scale = m_y_axis_scale_factor / 2;

    if (decrease_y_axis_scale >= 1)
        {
            m_y_axis_scale_factor = m_y_axis_scale_factor / 2;
            UpdateData(FALSE);      // Update the screen
            y_axis_scale_factor = m_y_axis_scale_factor;
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnIncreaseYAxisPositionButton()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////

    increase_y_axis_position_offset = m_y_axis_position_offset + 50;

    if (increase_y_axis_position_offset <= 10000)
        {
            m_y_axis_position_offset = m_y_axis_position_offset + 50;
            UpdateData(FALSE);      // Update the screen
            y_axis_position_offset = m_y_axis_position_offset;
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnDecreaseYAxisPositionButton()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////

    decrease_y_axis_position_offset = m_y_axis_position_offset - 50;

    if (decrease_y_axis_position_offset >= -10000)
```

```
        {
            m_y_axis_position_offset = m_y_axis_position_offset - 50;
            UpdateData(FALSE);      // Update the screen
            y_axis_position_offset = m_y_axis_position_offset;
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}

void CAutomaticDlg::OnIncreaseXAxisScaleButton()
{
    // TODO: Add your control notification handler code here

/////////////////////////////////////////
// MY CODE STARTS HERE
////////

    increase_x_axis_scale = m_x_axis_scale_factor * 2;

    if (increase_x_axis_scale <= 1000000)
        {
            m_x_axis_scale_factor = m_x_axis_scale_factor * 2;
            UpdateData(FALSE);      // Update the screen
            x_axis_scale_factor = m_x_axis_scale_factor;
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}

void CAutomaticDlg::OnDecreaseXAxisScaleButton()
{
    // TODO: Add your control notification handler code here

/////////////////////////////////////////
// MY CODE STARTS HERE
////////

    decrease_x_axis_scale = m_x_axis_scale_factor / 2;

    if (decrease_x_axis_scale >= 1)
        {
            m_x_axis_scale_factor = m_x_axis_scale_factor / 2;
            UpdateData(FALSE);      // Update the screen
            x_axis_scale_factor = m_x_axis_scale_factor;
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}

void CAutomaticDlg::OnIncreaseXAxisStarButton()
{
    // TODO: Add your control notification handler code here

/////////////////////////////////////////
// MY CODE STARTS HERE
////////
int     factor_to_be_increased_x_axis;
int     increase_x_axis_display_star_factor;

    factor_to_be_increased_x_axis =
                (2 * x_axis_display_star_constant * x_axis_percentage_constant)
                /(10 * m_x_axis_scale_factor);

    increase_x_axis_display_star_factor =
        m_x_axis_display_star_factor + factor_to_be_increased_x_axis;

    if (increase_x_axis_display_star_factor <= 9999)
        {
            m_x_axis_display_star_factor =
                    m_x_axis_display_star_factor + factor_to_be_increased_x_axis;
            UpdateData(FALSE);      // Update the screen
            x_axis_display_star_factor = m_x_axis_display_star_factor;
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
/////////////////////////////////////////
}

void CAutomaticDlg::OnDecreaseXAxisStarButton()
{
    // TODO: Add your control notification handler code here
```

```
//////////////////////////////////////////
// MY CODE STARTS HERE
////////
int     factor_to_be_decreased_x_axis;
int     decrease_x_axis_display_star_factor;

    factor_to_be_decreased_x_axis =
                (2 * x_axis_display_star_constant * x_axis_percentage_constant)
                /(10 * m_x_axis_scale_factor);

    decrease_x_axis_display_star_factor =
        m_x_axis_display_star_factor - factor_to_be_decreased_x_axis;

    if (decrease_x_axis_display_star_factor >= 0)
        {
            m_x_axis_display_star_factor =
                    m_x_axis_display_star_factor - factor_to_be_decreased_x_axis;
            UpdateData(FALSE);      // Update the screen
            x_axis_display_star_factor = m_x_axis_display_star_factor;
            Invalidate();   // Cause the execution of the OnPaint() function
        }
    else
        {
            m_x_axis_display_star_factor = 0;
            UpdateData(FALSE);      // Update the screen
            x_axis_display_star_factor = m_x_axis_display_star_factor;
            Invalidate();   // Cause the execution of the OnPaint() function
        }
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
//////////////          detection functions          ///////////////
//////////////          detection functions          ///////////////
//////////////          detection functions          ///////////////

//////////////////////////////////////////
// MY CODE STARTS HERE
////////
void initialising_detection_paramers()
{


    int counter_intialise_element;

    for(counter_intialise_element = 0;
        counter_intialise_element <= (data_section_total_point_no - 1);
        counter_intialise_element++)
        {
            real_data_array[counter_intialise_element].find_event_tempt_result = 0;
            real_data_array[counter_intialise_element].find_event_result = 0;
            real_data_array[counter_intialise_element].find_decay_time_result = 0;
            real_data_array[counter_intialise_element].find_rise_time_result = 0;
            real_data_array[counter_intialise_element].find_proximal_rise_time_result
= 0;

            real_data_array[counter_intialise_element
                                        ].test_belonged_to_event_result = 0;
            real_data_array[counter_intialise_element
                                        ].overlapping_previouse_event_result = 0;
            real_data_array[counter_intialise_element
                                        ].overlapping_foward_event_result = 0;

            real_data_array[counter_intialise_element].
                                        find_event_forward_limit_result = 0;


        }

}
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////




void CAutomaticDlg::OnDetectButton()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
```

```
// MY CODE STARTS HERE
////////

void initialising_detection_paramers();

void detect_data_section(REAL_DATA_TYPE *, int);

    initialising_detection_paramers();
    detect_data_section(real_data_array, data_section_total_point_no);

    Invalidate();   // Cause the execution of the OnPaint() function

//////////
// MY CODE ENDS HERE
////////////////////////////////////////////
}


////////////////////////////////////////
// MY CODE STARTS HERE
///a function to detect the event within a data section
void detect_data_section(REAL_DATA_TYPE *pointer_real_data_section_array,
                         int array_limit_no)
{
    int     detect_data_window(REAL_DATA_TYPE *, int, double, double);

    double  passed_window_width = 0.05;
    ///// with unit defined in the file, therefore, second

    int     passed_arrayed_element_no;

    int     last_detect_data_window_start_point_no;
    /// the point no within the data section

    int     counter_set_detect_data_window;


     //number of elements of the array passed equal to 50ms
    passed_arrayed_element_no =
        (int)((passed_window_width / pointer_real_data_section_array->time) *
                1.000001);

    last_detect_data_window_start_point_no =
        array_limit_no - passed_arrayed_element_no;


    for (counter_set_detect_data_window = 0;
        counter_set_detect_data_window <= last_detect_data_window_start_point_no;
        counter_set_detect_data_window++)
        {
            detect_data_window((pointer_real_data_section_array +
                                counter_set_detect_data_window),
                               passed_arrayed_element_no,
                                (pointer_real_data_section_array->time),
                               passed_window_width);
        }


    int confirm_event(REAL_DATA_TYPE *);

    int counter_finalise_event;

    for (counter_finalise_event = 0;
        counter_finalise_event < array_limit_no;
        counter_finalise_event++)
        {
            if( ((pointer_real_data_section_array +
                                counter_finalise_event)->find_event_tempt_result == 1)
                    &&
                    (confirm_event(pointer_real_data_section_array +
                                   counter_finalise_event) == 1) )
                {
                    (pointer_real_data_section_array +
                            counter_finalise_event)->find_event_result = 1;
                }
        }


    int test_event_overcounted(REAL_DATA_TYPE *);

    for (counter_finalise_event = array_limit_no - 1;
        counter_finalise_event >= 0;
        counter_finalise_event--)
        {
            if( ((pointer_real_data_section_array +
                                counter_finalise_event)->find_event_result == 1)
                    &&
```

```
                    (test_event_overcounted(pointer_real_data_section_array +
                                counter_finalise_event) == 1) )
                {
                    (pointer_real_data_section_array +
                            counter_finalise_event)->find_event_result = 0;
                }
        }


////////////////////////////////////////////////////////////////////
//////////////////////////define event and find parameters //////////////////////
void define_event_backward_range(REAL_DATA_TYPE *);
void define_event_foward_range(REAL_DATA_TYPE *);
void define_fit_decay_star_point(REAL_DATA_TYPE *);
int test_overlapping_previouse_event(REAL_DATA_TYPE *);


REAL_DATA_TYPE *pointer_fn_find_decay_time(REAL_DATA_TYPE *);
REAL_DATA_TYPE *pointer_fn_find_rise_time(REAL_DATA_TYPE *);
void find_decay_time_constant(REAL_DATA_TYPE *);


int time_to_no_of_point_ver2(double);

    double ending_time_tail = 0.01;
    //time in second that is reserved to measure the decay time

    int counter;

    for(counter = 0;
        counter <= (array_limit_no - 1
                    - time_to_no_of_point_ver2(ending_time_tail));
        counter++)
        {
            if ((pointer_real_data_section_array + counter)->find_event_result == 1)
                {
                    define_event_backward_range(pointer_real_data_section_array +
                                                counter);
                }
        }


    for(counter = 0;
        counter <= (array_limit_no - 1
                    - time_to_no_of_point_ver2(ending_time_tail));
        counter++)
        {
            if ((pointer_real_data_section_array + counter)->find_event_result == 1)
                {
                    define_event_foward_range(pointer_real_data_section_array +
                                                counter);
                }
        }


    for(counter = 0;
        counter <= (array_limit_no - 1
                    - time_to_no_of_point_ver2(ending_time_tail));
        counter++)
        {
            if ((pointer_real_data_section_array + counter)->find_event_result == 1)
                {
                    define_fit_decay_star_point(pointer_real_data_section_array +
                                                counter);
                }
        }




    for(counter = 0;
        counter <= (array_limit_no - 1
                    - time_to_no_of_point_ver2(ending_time_tail));
        counter++)
        {
            if ((pointer_real_data_section_array + counter)->find_event_result == 1)
                {
                    test_overlapping_previouse_event(pointer_real_data_section_array +

                                                counter);
                }
        }



    for(counter = 0;
        counter <= (array_limit_no - 1
                    - time_to_no_of_point_ver2(ending_time_tail));
```

```
        counter++)
        {
            if ((pointer_real_data_section_array + counter)->find_event_result == 1)
                {

                    pointer_fn_find_decay_time(pointer_real_data_section_array +
                                                counter);
                }
        }


    for(counter = 0;
        counter <= (array_limit_no - 1
                    - time_to_no_of_point_ver2(ending_time_tail));
        counter++)
        {
            if ((pointer_real_data_section_array + counter)->find_event_result == 1)
                {
                    pointer_fn_find_rise_time(pointer_real_data_section_array +
                                                counter);
                }
        }



    for(counter = 0;
        counter <= (array_limit_no - 1
                    - time_to_no_of_point_ver2(ending_time_tail));
        counter++)
        {
            if (
                ( (pointer_real_data_section_array + counter)->
                                                    find_event_result == 1 )
                &&
                ( (pointer_real_data_section_array + counter)->
                                            overlapping_previouse_event_result == 0)
                &&
                ( (pointer_real_data_section_array + counter)->
                                            overlapping_foward_event_result == 0)
                &&
                ( (pointer_real_data_section_array + counter)->
                                            find_event_forward_limit_result == 1)
                )
                {
                    find_decay_time_constant(pointer_real_data_section_array +
                                                counter);

                }
        }

//////////////////////////////define event and find parameters ///////////////////////
////////////////////////////////////////////////////////////////////////////

}

///a function to detect the event within a data section
// MY CODE ENDS HERE
/////////////////////////////////////////////


/////////////////////////////////////////
// MY CODE STARTS HERE
///a function to detect the event within a data window
//if an event is found return 1, otherwise return 0
int detect_data_window(REAL_DATA_TYPE *pointer_real_data_window_array,
                       int array_limit_no,
                       double time_interval,
                       double window_time_width)
{
    int find_event_result = 0;

    int time_to_no_of_point(double, double);
    int find_rising_phase(REAL_DATA_TYPE *);
    int find_false_rising_phase(REAL_DATA_TYPE *);


////////////////////////////////////////////////////////////////////////////
//////////////find the tempt peak//////////////////////////////
    REAL_DATA_TYPE *pointer_fn_find_tempt_peak(REAL_DATA_TYPE *, int);

    double time_range_to_find_temp_peak = 0.005;    //Unit: s

    double star_time_find_temp_peak = 0.02;  //Unit: s
    REAL_DATA_TYPE *pointer_star_find_temp_peak;

    pointer_star_find_temp_peak =
        (pointer_real_data_window_array +
```

```
        time_to_no_of_point(star_time_find_temp_peak, time_interval));

    REAL_DATA_TYPE *pointer_tempt_peak;

    pointer_tempt_peak =
        pointer_fn_find_tempt_peak(pointer_star_find_temp_peak,
        time_to_no_of_point(time_range_to_find_temp_peak, time_interval));
//////////////find the tempt peak//////////////////////////////
////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////
//////////////find the averaged base line current//////////////////////////////
    double averaged_base_line_current(REAL_DATA_TYPE *, int);
    double star_time_caculate_base_line = 0.01;  //Unit: s
    double time_range_to_caculate_base_line = 0.005;    //Unit: s


    REAL_DATA_TYPE *pointer_start_caculate_base_line;

    pointer_start_caculate_base_line =
        (pointer_real_data_window_array +
        time_to_no_of_point(star_time_caculate_base_line, time_interval));


    double base_line_current;   //Unit: s

    base_line_current =
        averaged_base_line_current(pointer_start_caculate_base_line,
        time_to_no_of_point(time_range_to_caculate_base_line, time_interval));



            pointer_tempt_peak->base_line_current = base_line_current;
            pointer_tempt_peak->amplitude =
                (base_line_current - pointer_tempt_peak->averaged_current);

//////////////find the averaged base line current//////////////////////////////
////////////////////////////////////////////////////////////////////////




////////////////////////////////////////////////////////////////////////
//////////////////////////////test the falling phase/////////
    int find_falling_phase(REAL_DATA_TYPE *);
    double time_gap_to_test_falling = 0.0025;  //Unit: s

    REAL_DATA_TYPE *pointer_start_test_falling_phase;

    pointer_start_test_falling_phase =
        (pointer_tempt_peak +
        time_to_no_of_point(time_gap_to_test_falling, time_interval));
//////////////////////////////test the falling phase/////////
////////////////////////////////////////////////////////////////////////

int find_false_amplitude(REAL_DATA_TYPE *);
int find_false_falling_phase(REAL_DATA_TYPE *);


////////////////////////////////////////////////////////////////////////
//////////////locate the tempt event/////////////////////////////////////

    double threshold_amplitude; /// unit nA

    threshold_amplitude = global_threshold_amplitude;

    if (
        (base_line_current - pointer_tempt_peak->averaged_current >=
        threshold_amplitude)
        &&
        (find_false_amplitude(pointer_tempt_peak) == 0)
        &&
        (find_rising_phase(pointer_tempt_peak) == 1)
        &&
        (find_false_rising_phase(pointer_tempt_peak) == 0)
        &&
        (find_falling_phase(pointer_start_test_falling_phase) == 1)
        &&
        (find_false_falling_phase(pointer_start_test_falling_phase) == 0)
        )
        {
            pointer_tempt_peak->find_event_tempt_result = 1;
            pointer_tempt_peak->base_line_current = base_line_current;
            pointer_tempt_peak->amplitude =
```

```
                (base_line_current - pointer_tempt_peak->averaged_current);
            find_event_result = 1;
        }
//////////////locate the tempt event//////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

    return (find_event_result);

}
///a function to detect the event within a data window
// MY CODE ENDS HERE
/////////////////////////////////////////


////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to find the tempt peak
// it return a pointer to the tempt peak point
REAL_DATA_TYPE *pointer_fn_find_tempt_peak(REAL_DATA_TYPE *pointer_test_point,
                                           int array_limit)
{
    REAL_DATA_TYPE *pointer_to_tempt_peak;

    pointer_to_tempt_peak = pointer_test_point;

    int counter;

    for(counter = 0;
        counter < array_limit;
        counter++)
        {
            if ((pointer_test_point + counter)->averaged_current
                        < pointer_to_tempt_peak->averaged_current)
                {
                    pointer_to_tempt_peak = (pointer_test_point + counter);
                }
        }

    return (pointer_to_tempt_peak);
}
///a function to find the tempt peak
// it return a pointer to the tempt peak point
// MY CODE ENDS HERE
/////////////////////////////////////////////


////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to find the averaged base line current
double averaged_base_line_current(REAL_DATA_TYPE *pointer_test_point,
                                  int array_limit)
{
    double averaged_base_line_current;

    double total_current = 0;

    int counter;

    for(counter = 0;
        counter < array_limit;
        counter++)
        {
            total_current =
                total_current + (pointer_test_point + counter)->current;
        }

    averaged_base_line_current = total_current / (double)array_limit;

    return (averaged_base_line_current);
}
///a function to find the averaged base line current
// MY CODE ENDS HERE
/////////////////////////////////////////


////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether a false amplitude exit in the data window
int find_false_amplitude(REAL_DATA_TYPE *pointer_star_point)
{

    int find_false_amplitude_result = 0;
    int time_to_no_of_point(double, double);
    int test_below_threshold_false_amplitude(REAL_DATA_TYPE *, REAL_DATA_TYPE *);

    double time_range_to_test_amplitude = 0.001;    //Unit: s
```

```
    int test_false_amplitude_result = 0;
    double time_proportion = 0.2;
    /// out of the time_range_to_test_false_amplitude,
    // if a time_proportion have amplitude smaller than the
    /// threshold amplitude, the amplitude is false

    int counter;

    for(counter = 0;
        counter <= time_to_no_of_point(time_range_to_test_amplitude,
                                       pointer_star_point->time_interval);
        counter++)
        {
            test_false_amplitude_result =
                           test_false_amplitude_result +
                           test_below_threshold_false_amplitude(
                                   pointer_star_point,
                                   (pointer_star_point +
                                   counter));
        }

    if (test_false_amplitude_result >
        time_to_no_of_point( (time_range_to_test_amplitude * time_proportion),
                           pointer_star_point->time_interval))
        {   ////therefore false falling phase is found
            find_false_amplitude_result = 1;
        }

    return (find_false_amplitude_result);

}
///a function to test whether a false amplitude exit in the data window
// MY CODE ENDS HERE
////////////////////////////////////////////


////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether the given point exceed the threshold
//  false amplitude
int test_below_threshold_false_amplitude(REAL_DATA_TYPE *pointer_reference_point,
                                         REAL_DATA_TYPE *pointer_test_point)
{
    int test_below_threshold_result = 0;

    double threshold_amplitude;  //Unit: nA/s
    double amplitude_factor = 0.5;
    threshold_amplitude = pointer_reference_point->amplitude;
    /// if a false falling amplitude is detect, it should has 0.001s
    /// with amplitude below the threshold_amplitude


    if ( (pointer_reference_point->base_line_current -
          pointer_test_point->current)
        <=
        threshold_amplitude * amplitude_factor)
        {
            test_below_threshold_result = 1;
        }

    return (test_below_threshold_result);
}
///a function to test whether the given point exceed the threshold
//  false amplitude
// MY CODE ENDS HERE
////////////////////////////////////////////




////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether a rising phase exit in the data window
int find_rising_phase(REAL_DATA_TYPE *pointer_star_point)
{
    int find_rising_phase_result = 0;
    int test_rise_slope(REAL_DATA_TYPE *);
    int time_to_no_of_point(double, double);

    double time_range_to_test_rising = 0.004;   //Unit: s

    int test_rising_phase_result = 0;
    int test_rising_phase_threshold;
    test_rising_phase_threshold = global_rising_threshold_point_no;
```

```
    /// if a rising phase is detect, it should greater than
    /// test_rising_phase_threshold

    int counter_test_rising_phase;

    for(counter_test_rising_phase =
            ( -time_to_no_of_point(time_range_to_test_rising,
                    pointer_star_point->time_interval) );
            counter_test_rising_phase <= 0;
            counter_test_rising_phase++)
        {
            test_rising_phase_result =
                            test_rising_phase_result +
                            test_rise_slope((pointer_star_point +
                            counter_test_rising_phase));
        }

    if (test_rising_phase_result >= test_rising_phase_threshold)
        {   ////therefore, rising phase is found
            find_rising_phase_result = 1;
        }

    return (find_rising_phase_result);
}
///a function to test whether a rising phase exit in the data window
// MY CODE ENDS HERE
////////////////////////////////////////////



////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether the given point is in a rising slope
int test_rise_slope(REAL_DATA_TYPE *pointer_test_point)
{

    int test_rising_result = 0;
    double slope;
    double threshold_slope;         //unit: nA per s

    threshold_slope = global_rising_threshold_slope;

    slope = ((pointer_test_point + 1)->current - pointer_test_point->current)
            / pointer_test_point->time_interval;

    if (slope <= threshold_slope)
        {
            test_rising_result = 1;
        }

    return (test_rising_result);
}
///a function to test whether the given point is in a rising slope
// MY CODE ENDS HERE
////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether a false rising phase exit in the data window
int find_false_rising_phase(REAL_DATA_TYPE *pointer_star_point)
{
    int find_false_rising_phase_result = 0;
    int time_to_no_of_point(double, double);
    int test_false_rise_slope(REAL_DATA_TYPE *);

    double time_range_to_test_false_rising = 0.002; //Unit: s

    int test_false_rising_phase_result = 0;
    int test_false_rising_phase_threshold = 2;
    /// if a rising phase is detect to be false, it should have
    /// test_false_rising_phase_result greater than
    /// test_false_rising_phase_threshold

    int counter_test_false_rising_phase;

    for(counter_test_false_rising_phase =
                ( -time_to_no_of_point(time_range_to_test_false_rising,
                pointer_star_point->time_interval) );
        counter_test_false_rising_phase <= 0;
        counter_test_false_rising_phase++)
        {
            test_false_rising_phase_result =
                        test_false_rising_phase_result +
                        test_false_rise_slope((pointer_star_point +
                        counter_test_false_rising_phase));
        }
```

```
    if (test_false_rising_phase_result >= test_false_rising_phase_threshold)
        {   ////therefore, the rising phase is false
            find_false_rising_phase_result = 1;
        }

    return (find_false_rising_phase_result);

}
///a function to test whether a false rising phase exit in the data window
// MY CODE ENDS HERE
//////////////////////////////////////////


//////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether the given point is in a false rising slope
int test_false_rise_slope(REAL_DATA_TYPE *pointer_test_point)
{
    int test_false_rising_result = 0;
    double slope;
    double threshold_slope = 100;   //unit: nA per s

    slope = ((pointer_test_point + 1)->current - pointer_test_point->current)
            / pointer_test_point->time_interval;

    if (slope >= threshold_slope)
        {
            test_false_rising_result = 1;
        }

    return (test_false_rising_result);

}
///a function to test whether the given point is in a false rising slope
// MY CODE ENDS HERE
//////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether a falling phase exit in the data window
int find_falling_phase(REAL_DATA_TYPE *pointer_star_point)
{

    int find_falling_phase_result = 0;
    int time_to_no_of_point(double, double);
    int test_exceed_threshold_falling_slope(REAL_DATA_TYPE *);

    double time_range_to_test_falling = 0.003;  //Unit: s

    int test_falling_phase_result = 0;
    double time_proportion;
    time_proportion = global_falling_time_proportion;
    /// out of the time_range_to_test_falling, at least
    // a time_proportion should
    /// have slope exeed the threshold slope

    int counter_test_falling_phase;

    for(counter_test_falling_phase = 0;
        counter_test_falling_phase <=
                time_to_no_of_point(time_range_to_test_falling,
                                    pointer_star_point->time_interval);
        counter_test_falling_phase++)
        {
            test_falling_phase_result =
                            test_falling_phase_result +
                            test_exceed_threshold_falling_slope((
                                    pointer_star_point +
                                    counter_test_falling_phase));
        }

    if (test_falling_phase_result >=
        time_to_no_of_point( (time_range_to_test_falling * time_proportion),
                            pointer_star_point->time_interval))
        {   ////therefore falling phase is found
            find_falling_phase_result = 1;
        }

    return (find_falling_phase_result);

}
///a function to test whether a falling phase exit in the data window
// MY CODE ENDS HERE
//////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether the given point exceed the threshold
//  fallin slope
int test_exceed_threshold_falling_slope(REAL_DATA_TYPE *pointer_test_point)
{
    int test_exceed_threshold_result = 0;

    double threshold_slope;  //Unit: nA/s
    threshold_slope = global_falling_threshold_slope;
    /// if a falling phase is detect, it should has 0.003s
    /// with general slope greater than the threshold slope


    if (pointer_test_point->general_slope >= threshold_slope)
        {
            test_exceed_threshold_result = 1;
        }

    return (test_exceed_threshold_result);
}
///a function to test whether the given point exceed the threshold
//  fallin slope
// MY CODE ENDS HERE
////////////////////////////////////////////



////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether a false falling phase exit in the data window
int find_false_falling_phase(REAL_DATA_TYPE *pointer_star_point)
{

    int find_false_falling_phase_result = 0;
    int time_to_no_of_point(double, double);
    int test_exceed_threshold_false_falling_slope(REAL_DATA_TYPE *);

    double time_range_to_test_falling = 0.003;  //Unit: s

    int test_false_falling_phase_result = 0;
    double time_proportion = 0.6;
    /// out of the time_range_to_test_false_falling, at least
    // a time_proportion should
    /// have slope exeed the threshold slope

    int counter;

    for(counter = 0;
        counter <= time_to_no_of_point(time_range_to_test_falling,
                                       pointer_star_point->time_interval);
        counter++)
        {
            test_false_falling_phase_result =
                            test_false_falling_phase_result +
                            test_exceed_threshold_false_falling_slope((
                                    pointer_star_point +
                                    counter));
        }

    if (test_false_falling_phase_result >=
        time_to_no_of_point( (time_range_to_test_falling * time_proportion),
                            pointer_star_point->time_interval))
        {   ////therefore false falling phase is found
            find_false_falling_phase_result = 1;
        }

    return (find_false_falling_phase_result);

}
///a function to test whether a false falling phase exit in the data window
// MY CODE ENDS HERE
////////////////////////////////////////////



////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether the given point exceed the threshold
//  false falling slope
int test_exceed_threshold_false_falling_slope(REAL_DATA_TYPE *pointer_test_point)
{
    int test_exceed_threshold_result = 0;

    double threshold_slope;  //Unit: nA/s
```

```
    threshold_slope = 0.0001;
    /// if a false falling phase is detect, it should has 0.003s
    /// with general slope greater than the threshold slope


    if (pointer_test_point->general_slope <= threshold_slope)
        {
            test_exceed_threshold_result = 1;
        }

    return (test_exceed_threshold_result);
}
///a function to test whether the given point exceed the threshold
//  false falling slope
// MY CODE ENDS HERE
////////////////////////////////////////////




////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to confirm a tempt event is a true event
int confirm_event(REAL_DATA_TYPE *pointer_test_point)
{
    int time_to_no_of_point(double, double);
    int confirm_result = 1;

    double half_time_interval = 0.005; ////Unit: s

    int counter;

    for(counter = (-time_to_no_of_point(half_time_interval,
                                        pointer_test_point->time_interval));
        counter <= 0;
        counter++)
        {
            if ((pointer_test_point + counter)->averaged_current
                        < pointer_test_point->averaged_current)
                {
                    confirm_result = 0;
                }
        }


    for(counter = 0;
        counter <= time_to_no_of_point(half_time_interval,
                                        pointer_test_point->time_interval);
        counter++)
        {
            if ((pointer_test_point + counter)->averaged_current
                        < pointer_test_point->averaged_current)
                {
                    confirm_result = 0;
                }
        }

    return (confirm_result);
}
///a function to confirm a tempt event is a true event
// MY CODE ENDS HERE
////////////////////////////////////////////



////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to confirm a tempt event is a true event
int test_event_overcounted(REAL_DATA_TYPE *pointer_test_point)
{
    int time_to_no_of_point(double, double);
    int overcounted_value = 0;

    double half_time_interval = 0.005; ////Unit: s

    int counter;

    for(counter = (-time_to_no_of_point(half_time_interval,
                                        pointer_test_point->time_interval));
        counter < 0;
        counter++)
        {
            if ((pointer_test_point + counter)->find_event_result == 1)
                {
                    overcounted_value = 1;
                }
        }
```

```
    return (overcounted_value);
}
///a function to confirm a tempt event is a true event
// MY CODE ENDS HERE
//////////////////////////////////////////


//////////////          detection functions          ///////////////
//////////////          detection functions          ///////////////
//////////////          detection functions          ///////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
//////////////            define event                ///////////////
//////////////            define event                ///////////////
//////////////            define event                ///////////////



//////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to find the backward range of an event
void define_event_backward_range(REAL_DATA_TYPE *pointer_test_point)
{
int time_to_no_of_point_ver2(double time);


    double rise_time_proportion;
           rise_time_proportion = 0.1;

    double time_range_searching = 0.01;


    pointer_test_point->pointer_to_event_backward_limit = pointer_test_point;

    int counter;

    for(counter = -1;
        counter > (-time_to_no_of_point_ver2(time_range_searching));
        counter--)
        {
            if (
                ((pointer_test_point + counter)->find_event_result == 1)
                ||
                ((pointer_test_point->base_line_current -
                 (pointer_test_point + counter)->averaged_current)
                 <=
                 (pointer_test_point->amplitude * rise_time_proportion))
                 )
                {
                    pointer_test_point->pointer_to_event_backward_limit
                        = (pointer_test_point + counter + 1);
                    /////define the event backward limit

                    break;
                }

            (pointer_test_point + counter)->test_belonged_to_event_result = 1;

            (pointer_test_point + counter)->pointer_to_belonged_event
                = pointer_test_point;

        }

}
///a function to find the backward range of an event
// MY CODE ENDS HERE
//////////////////////////////////////////


//////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to find the forward range of an event
void define_event_foward_range(REAL_DATA_TYPE *pointer_test_point)
{
int time_to_no_of_point_ver2(double time);


    double decay_time_proportion;
           decay_time_proportion = 0.1;
```

```
    double time_range_searching = 0.08;

    pointer_test_point->pointer_to_event_forward_limit = pointer_test_point;


    int counter;

    for(counter = 1;
        counter < time_to_no_of_point_ver2(time_range_searching);
        counter++)
        {
            if (
                    (
                ( (pointer_test_point + counter)->test_belonged_to_event_result == 1 )
                &&
                (     (   (REAL_DATA_TYPE *)( (pointer_test_point +
                                counter)->pointer_to_belonged_event )    )
                != pointer_test_point     )
                    )
                        ||
                    (
                (pointer_test_point->base_line_current -
                 (pointer_test_point + counter)->averaged_current)
                 <=
                 (pointer_test_point->amplitude * decay_time_proportion)
                    )
                        ||
                    (
                (pointer_test_point + counter) >=
                        &real_data_array[data_section_total_point_no - 1]
                    )
                 )
                {
                    pointer_test_point->pointer_to_event_forward_limit
                        = (pointer_test_point + counter - 1);
                    /////define the event forward limit
                    pointer_test_point->find_event_forward_limit_result = 1;
                    /////define the event forward limit
                    break;
                }

            (pointer_test_point + counter)->test_belonged_to_event_result = 1;

            (pointer_test_point + counter)->pointer_to_belonged_event
                = pointer_test_point;

            (pointer_test_point + counter)->event_local_time
                = (pointer_test_point + counter)->time - pointer_test_point->time;

        }



}
///a function to find the forward range of an event
// MY CODE ENDS HERE
//////////////////////////////////////////////




//////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to define the beginnig of fitting decay
void define_fit_decay_star_point(REAL_DATA_TYPE *pointer_to_event)
{
int time_to_no_of_point_ver2(double time);


    double begin_proportion;
           begin_proportion = 0.9;

    double time_range_searching = 0.08;

    pointer_to_event->pointer_to_begin_fit_decay = pointer_to_event;


    int counter;

    for(counter = 1;
        counter < time_to_no_of_point_ver2(time_range_searching);
        counter++)
        {
            if (
                    (
```

```
                ( (pointer_to_event + counter)->test_belonged_to_event_result == 1 )
                &&
                (     (   (REAL_DATA_TYPE *)( (pointer_to_event +
                                counter)->pointer_to_belonged_event )    )
                != pointer_to_event     )
                    )
                        ||
                    (
                (pointer_to_event->base_line_current -
                 (pointer_to_event + counter)->averaged_current)
                 <=
                 (pointer_to_event->amplitude * begin_proportion)
                    )
                        ||
                    (
                (pointer_to_event + counter) >=
                        &real_data_array[data_section_total_point_no - 1]
                    )
                 )
                {
                    pointer_to_event->pointer_to_begin_fit_decay
                        = (pointer_to_event + counter - 1);
                    /////define the the beginning of fitting decay
                    break;
                }

        }


}
///a function to define the beginnig of fitting decay
// MY CODE ENDS HERE
//////////////////////////////////////////


////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to test whether an event overlapping with previouse event
int test_overlapping_previouse_event(REAL_DATA_TYPE *pointer_test_point)
{
    int time_to_no_of_point(double, double);
    int overlapping_value = 0;

    double time_testing_overlap = 0.02; ////Unit: s

    int counter;

    for(counter = -1;
        counter >= (-time_to_no_of_point(time_testing_overlap,
                                pointer_test_point->time_interval));
        counter--)
        {

            if (
                ((pointer_test_point + counter)->test_belonged_to_event_result == 1)
                &&
                ( ((REAL_DATA_TYPE *)(pointer_test_point +
                                counter)->pointer_to_belonged_event)
                != pointer_test_point)
                )
                {
                    overlapping_value = 1;
                    pointer_test_point->overlapping_previouse_event_result = 1;

                    pointer_test_point->pointer_to_overlapping_previouse_event
                        = ((REAL_DATA_TYPE *)(pointer_test_point +
                                counter)->pointer_to_belonged_event);



                    ((REAL_DATA_TYPE *)(pointer_test_point +
                                counter)->pointer_to_belonged_event)->
                                overlapping_foward_event_result = 1;

                    ((REAL_DATA_TYPE *)(pointer_test_point +
                                counter)->pointer_to_belonged_event)->
                                pointer_to_overlapping_foward_event
                    = pointer_test_point;

                    break;
```

```
            }

        }

    return (overlapping_value);
}
///a function to test whether an event overlapping with previouse event
// MY CODE ENDS HERE
//////////////////////////////////////////



//////////////          define event                ///////////////
//////////////          define event                ///////////////
//////////////          define event                ///////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////////        find event parameters         ///////////////
//////////////        find event parameters         ///////////////
//////////////        find event parameters         ///////////////


//////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to find the decay time
// it return a pointer to the decay time point
REAL_DATA_TYPE *pointer_fn_find_decay_time(REAL_DATA_TYPE *pointer_test_point)
{
int time_to_no_of_point_ver2(double time);
    REAL_DATA_TYPE *pointer_to_decay_time;

    pointer_to_decay_time = pointer_test_point;
    pointer_test_point->pointer_to_decay_time = pointer_test_point;

    double decay_time;
           decay_time = global_decay_time;

    double time_range_searching_decay_time = 0.05;

    int counter;

    for(counter = 1;
        counter < time_to_no_of_point_ver2(time_range_searching_decay_time);
        counter++)
        {
            if (
                    (
                ((pointer_test_point + counter)->test_belonged_to_event_result == 1)
                &&
                ( ((REAL_DATA_TYPE *)((pointer_test_point +
                                       counter)->pointer_to_belonged_event))
                != pointer_test_point)
                    )
                    ||
                    (
                (pointer_test_point + counter) >=
                       &real_data_array[data_section_total_point_no - 1]
                    )
                )
                {
                    break;
                }



            if ( (pointer_test_point->base_line_current -
                  (pointer_test_point + counter)->averaged_current)
                 <=
                 (pointer_test_point->amplitude * decay_time))
                {
                    pointer_to_decay_time = (pointer_test_point + counter);
                    pointer_test_point->find_decay_time_result = 1;

                    pointer_test_point->pointer_to_decay_time
                        = (pointer_test_point + counter);
                    break;
                }
        }
```

```
////////////////////////////////////////////////////////////////////////////////
/////////////////correct the decay time ////////////////////////////////////
    double time_range_to_correct_decay_time = 0.002;

    REAL_DATA_TYPE *pointer_to_decay_time_corrected;

    pointer_to_decay_time_corrected = pointer_to_decay_time;


    REAL_DATA_TYPE *pointer_counter;

if ( pointer_test_point->find_decay_time_result == 1)
{ ////star if statement

    for(pointer_counter = (pointer_to_decay_time +
                time_to_no_of_point_ver2(time_range_to_correct_decay_time));
        pointer_counter > pointer_to_decay_time;
        pointer_counter--)
        {
            if (
                (
                (pointer_counter->test_belonged_to_event_result == 1)
                &&
                (   ( (REAL_DATA_TYPE *)(pointer_counter->pointer_to_belonged_event) )

                != pointer_test_point    )
                    )
                    ||
                    (
                pointer_counter >=
                        &real_data_array[data_section_total_point_no - 1]
                    )
                )
                {
                    break;
                }


            if ( (pointer_test_point->base_line_current -
                 pointer_counter->averaged_current)
                 >=
                 (pointer_test_point->amplitude * decay_time))
                {
                    pointer_to_decay_time_corrected = pointer_counter;
                    break;
                }
        }

        pointer_test_point->pointer_to_decay_time =
            pointer_to_decay_time +
            ((pointer_to_decay_time_corrected->index -
            pointer_to_decay_time->index) / 2);

} ////end if statement

/////////////////correct the decay time ////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

    if ( pointer_test_point->find_decay_time_result == 1)
        {
                pointer_test_point->decay_time
            =
                ( (REAL_DATA_TYPE *)(pointer_test_point->pointer_to_decay_time) )->tim
e
                -
                pointer_test_point->time;
        }

    return (pointer_to_decay_time);
}
///a function to find the decay time
// it return a pointer to the decay time point
// MY CODE ENDS HERE
////////////////////////////////////////////


////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to find the rise time
// it return a pointer to the rise time point
REAL_DATA_TYPE *pointer_fn_find_rise_time(REAL_DATA_TYPE *pointer_test_point)
{
int time_to_no_of_point_ver2(double time);
    REAL_DATA_TYPE *pointer_to_rise_time;

    pointer_to_rise_time = pointer_test_point;
```

```
    pointer_test_point->pointer_to_rise_time = pointer_test_point;
    pointer_test_point->pointer_to_proximal_rise_time = pointer_test_point;

    double rise_time_factor;
           rise_time_factor = 0.2;

    double time_range_searching_rise_time = 0.01;

    int counter;

    for(counter = 0;
        counter > (-time_to_no_of_point_ver2(time_range_searching_rise_time));
        counter--)
        {
            if ( (pointer_test_point->base_line_current -
                 (pointer_test_point + counter)->current)
                 <=
                 (pointer_test_point->amplitude * rise_time_factor))
                {
                    pointer_to_rise_time = (pointer_test_point + counter);
                    pointer_test_point->find_rise_time_result = 1;

                    pointer_test_point->pointer_to_rise_time =
                                    (pointer_test_point + counter);
                    break;
                }
        }



    double proximal_rise_time_factor;
           proximal_rise_time_factor = 0.8;

    for(counter = 0;
        counter > (-time_to_no_of_point_ver2(time_range_searching_rise_time));
        counter--)
        {
            if ( (pointer_test_point->base_line_current -
                 (pointer_test_point + counter)->current)
                 <=
                 (pointer_test_point->amplitude * proximal_rise_time_factor))
                {
                    pointer_test_point->find_proximal_rise_time_result = 1;

                    pointer_test_point->pointer_to_proximal_rise_time
                        = (pointer_test_point + counter);
                    break;
                }
        }


    if (
         (pointer_test_point->find_rise_time_result == 1)
         &&
         (pointer_test_point->find_proximal_rise_time_result == 1)
         )
         {
                pointer_test_point->rise_time
            =
                ( (REAL_DATA_TYPE *)(pointer_test_point->pointer_to_proximal_rise_time
) )->time
                -
                ( (REAL_DATA_TYPE *)(pointer_test_point->pointer_to_rise_time) )->time
;
         }

     return (pointer_to_rise_time);
}
///a function to find the rise time
// it return a pointer to the rise time point
// MY CODE ENDS HERE
////////////////////////////////////////////


////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to find the decay time constant of the event
void find_decay_time_constant(REAL_DATA_TYPE *pointer_to_event)
{
void set_ideal_decay_current(REAL_DATA_TYPE *, FITTED_DATA_TYPE);
double mean_current_error_square(REAL_DATA_TYPE *, FITTED_DATA_TYPE);
```

```
    double decay_time_constant;
    double minimum = 0.0001;   //unit: ms
    double maximum = 0.05;  //unit: ms
    double interval = 0.0001;  //unit: ms

    double least_error;
    double mean_error;
    FITTED_DATA_TYPE    parameter;

    double ideal_decay_time_constant = 0;


            parameter.decay_time_constant = minimum;
            least_error = mean_current_error_square(pointer_to_event, parameter);
            ////// initialise least_error

    for(decay_time_constant = minimum;
        decay_time_constant <= maximum;
        decay_time_constant = decay_time_constant + interval)
        {
            parameter.decay_time_constant = decay_time_constant;
            mean_error = mean_current_error_square(pointer_to_event, parameter);


            if (mean_error < least_error)
                {
                    least_error = mean_error;
                    ideal_decay_time_constant = decay_time_constant;
                }

        }



    pointer_to_event->mean_decay_current_error_2 = least_error;
    pointer_to_event->decay_time_constant = ideal_decay_time_constant;

    parameter.decay_time_constant = ideal_decay_time_constant;
    set_ideal_decay_current(pointer_to_event, parameter);
            ////// set the ideal_fitted_current of the event

}
///a function to find the decay time constant of the event
// MY CODE ENDS HERE
//////////////////////////////////////////



////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to set the ideal decay current of the event
void set_ideal_decay_current(REAL_DATA_TYPE *pointer_to_event,
                             FITTED_DATA_TYPE   ideal_decay)
{

    REAL_DATA_TYPE *pointer_counter;


    for(pointer_counter = (  (REAL_DATA_TYPE *)(pointer_to_event->
                                        pointer_to_begin_fit_decay)  );
        pointer_counter <= (  (REAL_DATA_TYPE *)(pointer_to_event->
                                        pointer_to_event_forward_limit)  );
        pointer_counter++)
        {

            pointer_counter-> ideal_fitted_current
                =
                pointer_to_event->base_line_current -
                (
                    pointer_to_event->amplitude *
                    exp(   -(pointer_counter->event_local_time /
                             ideal_decay.decay_time_constant)   )
                );

        }

}
///a function to set the ideal decay current of the event
// MY CODE ENDS HERE
//////////////////////////////////////////


////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to calculate the mean square error of decay fitting
double mean_current_error_square(REAL_DATA_TYPE *pointer_to_event,
```

```
                                FITTED_DATA_TYPE    parameter)
{

void set_ideal_decay_current(REAL_DATA_TYPE *, FITTED_DATA_TYPE);

    set_ideal_decay_current(pointer_to_event, parameter);


    double sum_of_error_square = 0;
    double mean_error_square;
    double number_of_data_point = 0;

    REAL_DATA_TYPE *pointer_counter;

    for(pointer_counter = (   (REAL_DATA_TYPE *)(pointer_to_event->
                                        pointer_to_begin_fit_decay)   );
        pointer_counter <= (   (REAL_DATA_TYPE *)(pointer_to_event->
                                        pointer_to_event_forward_limit)   );
        pointer_counter++)
        {

            sum_of_error_square
                =
                sum_of_error_square +
                pow((   pointer_counter->ideal_fitted_current -
                pointer_counter->current    ), 2);

            number_of_data_point = number_of_data_point + 1;

        }


    mean_error_square = sum_of_error_square / number_of_data_point;

    return (mean_error_square);

}
///a function to calculate the mean square error of decay fitting
// MY CODE ENDS HERE
////////////////////////////////////////////

//////////////          find event parameters          ///////////////
//////////////          find event parameters          ///////////////
//////////////          find event parameters          ///////////////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////
// MY CODE STARTS HERE
///a function to convert time to no of point
int time_to_no_of_point(double time, double time_interval)
{

    int no_of_point;
    no_of_point = (int)((time / time_interval) * 1.000001);
    return (no_of_point);
}

///a function to test whether the given point is in a rising slope
// MY CODE ENDS HERE
////////////////////////////////////////////

////////////////////////////////////////
// MY CODE STARTS HERE
///a function to convert time to no of point
int time_to_no_of_point_ver2(double time)
{

    int no_of_point;
    no_of_point = (int)((time / (double)GetDSChan_xScale) * 1.000001);
    return (no_of_point);
}

///a function to test whether the given point is in a rising slope
// MY CODE ENDS HERE
////////////////////////////////////////////



////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to found out whether the given point's slope is positve,
// negative or zero
int test_positive_negative_or_zero_slope(REAL_DATA_TYPE *pointer_test_point)
{
```

```
    int result = 0; // if the slope is positive, return 1
                    // if the slope is negative, return -1
                    // if the slope is zero, return 0
    double slope;

    slope = ((pointer_test_point + 1)->current - pointer_test_point->current)
            / pointer_test_point->time_interval;


    if (slope > 0)
        {
            return (1);
        }
    else
        {
            if (slope < 0)
                {
                    return (-1);
                }
        }

    return (result);
}
///a function to found out whether the given point's slope is positve,
// negative or zero
// MY CODE ENDS HERE
////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to found out the slope of the point
double slope_of_the_point(REAL_DATA_TYPE *pointer_test_point,
                          double time_interval)
{
    double slope;

    slope = ((pointer_test_point + 1)->current - pointer_test_point->current)
            / time_interval;

    return (slope);

}

///a function to found out the slope of the point
// MY CODE ENDS HERE
////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to found out the gerneral slope of the point,
// using 5 points to caculate the average current before computing the slope
double general_slope_of_the_point(REAL_DATA_TYPE *pointer_to_the_point,
                                  double time_interval)
{
    double slope;
    REAL_DATA_TYPE  *pointer_3_point_before;
    REAL_DATA_TYPE  *pointer_3_point_after;

    double average_current_before = 0;
    double average_current_after = 0;

    pointer_3_point_before = pointer_to_the_point - 3;
    pointer_3_point_after = pointer_to_the_point + 3;

    int counter_caculate_average;

    for (counter_caculate_average = -2;
        counter_caculate_average <= 2;
        counter_caculate_average++)
        {
            average_current_before =
                (average_current_before +
                (pointer_3_point_before + counter_caculate_average)->current);
        }
    average_current_before = average_current_before / 5;



    for (counter_caculate_average = -2;
        counter_caculate_average <= 2;
        counter_caculate_average++)
        {
            average_current_after =
              (average_current_after +
                (pointer_3_point_after + counter_caculate_average)->current);
```

```
        }
    average_current_after = average_current_after / 5;

    slope = (average_current_after - average_current_before) / (time_interval * 6);

    return (slope);

}

///a function to found out the gerneral slope of the point,
// using 5 points to caculate the average current before computing the slope
// MY CODE ENDS HERE
//////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to find the average current using 3 points
double averaging_current(REAL_DATA_TYPE *pointer_test_point)
{
    double averaged_current;

    averaged_current =
            ((pointer_test_point - 1)->current +
              pointer_test_point->current +
              (pointer_test_point + 1)->current)
            / 3;

    return (averaged_current);

}
///a function to find the average current using 3 points
// MY CODE ENDS HERE
//////////////////////////////////////////////////////////////////////////////

void CAutomaticDlg::OnKillfocusThresholdAmplitudeEdit()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    global_threshold_amplitude = m_threshold_amplitude;
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnKillfocusRisingThresholdSlopeEdit()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    global_rising_threshold_slope = m_rising_threshold_slope;
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnKillfocusFallingThresholdSlopeEdit()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    global_falling_threshold_slope = m_falling_threshold_slope;
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnKillfocusRisingThresholdPointNoEdit()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    global_rising_threshold_point_no = m_rising_threshold_point_no;
```

```
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnKillfocusFallingTimeProportionEdit()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    global_falling_time_proportion = m_falling_time_proportion;
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnKillfocusDecayTimeEdit()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////
    UpdateData(TRUE);   // Update the variables of the controls.
    global_decay_time = m_decay_time;
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnKillfocusFinalDetectedDataSectionNoEdit()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////
    long temp_m_final_detected_data_section_no;

    temp_m_final_detected_data_section_no = m_final_detected_data_section_no;
    UpdateData(TRUE);   // Update the variables of the controls.

    if (m_final_detected_data_section_no > file_data_section_no)
        {
            m_final_detected_data_section_no = temp_m_final_detected_data_section_no;
            UpdateData(FALSE);      // Update the screen
        }
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////
}

void CAutomaticDlg::OnDetectSectionsButton()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
////////
    void copy_event();

    begin_detect_data_section_no = m_current_data_section_no;
    final_detected_data_section_no = m_final_detected_data_section_no;

    pointer_write_event = event_array;
    pointer_write_total_event = total_event_array;

    written_event_no = 0;
    total_event_no = 0;
    overlapping_event_no = 0;

    total_time_for_detection = 0;

    while ( m_current_data_section_no < m_final_detected_data_section_no )
        {
            CAutomaticDlg::OnDetectButton();
            copy_event();

            total_time_for_detection =
                total_time_for_detection +
                (data_section_total_point_no * GetDSChan_xScale);

            CAutomaticDlg::OnDataSectionForwardButton();
        }
```

```
            CAutomaticDlg::OnDetectButton();
            copy_event();

            total_time_for_detection =
                total_time_for_detection +
                (data_section_total_point_no * GetDSChan_xScale);


    overlapping_event_proportion = (double)overlapping_event_no / (double)total_event_
no;

    total_event_frequency = (double)total_event_no / total_time_for_detection;

    no_of_data_section_for_detection =
            final_detected_data_section_no - begin_detect_data_section_no + 1;




////////////////////////////////////////////////////////////////////////////
//////////////set inter event interval of total_event_array///////////////
    double inter_event_interval(REAL_DATA_TYPE *);

    int counter;

    for(counter = 1;
        counter <= (total_event_no - 1);
        counter++)
        {
            total_event_array[counter].inter_event_interval =
            inter_event_interval(&total_event_array[counter]);
        }

//////////////set inter event interval of total_event_array///////////////
////////////////////////////////////////////////////////////////////////////



//////////
// MY CODE ENDS HERE
/////////////////////////////////////////////
}



/////////////////////////////////////////
// MY CODE STARTS HERE
////////
void copy_event()
{


    int counter;

    for(counter = 0;
        counter <= (data_section_total_point_no - 1);
        counter++)
        {
            if (
                (real_data_array[counter].find_event_result == 1)
                &&
                (real_data_array[counter].overlapping_previouse_event_result == 0)
                &&
                (real_data_array[counter].overlapping_foward_event_result == 0)
                &&
                (real_data_array[counter].find_decay_time_result == 1)
                &&
                (real_data_array[counter].find_rise_time_result == 1)
                &&
                (real_data_array[counter].find_proximal_rise_time_result == 1)
                &&
                (real_data_array[counter].find_event_forward_limit_result == 1)
                )
                {
                    *pointer_write_event = real_data_array[counter];
                    pointer_write_event = pointer_write_event + 1;

                    written_event_no = written_event_no + 1;
                }


            if (
                    (real_data_array[counter].find_event_result == 1)
                &&
                    (
```

```
                    (real_data_array[counter].overlapping_previouse_event_result == 1)
                    ||
                    (real_data_array[counter].overlapping_foward_event_result == 1)
                    )
                )
                {
                    overlapping_event_no = overlapping_event_no + 1;
                }


            if (real_data_array[counter].find_event_result == 1)
                {
                    *pointer_write_total_event = real_data_array[counter];
                    pointer_write_total_event = pointer_write_total_event + 1;

                    total_event_no = total_event_no + 1;
                }

        }

}
//////////
// MY CODE ENDS HERE
////////////////////////////////////////////



////////////////////////////////////////////////////////////
// MY CODE STARTS HERE
///a function to calculate the inter event interval
double inter_event_interval(REAL_DATA_TYPE *pointer_to_event)
{



    double interval;


    interval = (
                (   pointer_to_event->section_no - (pointer_to_event- 1)->section_no
 )
                * (double)GetDSChan_xScale * data_section_total_point_no
                )
                +
                (pointer_to_event->time - (pointer_to_event- 1)->time);

    return (interval);

}
///a function to calculate the inter event interval
// MY CODE ENDS HERE
////////////////////////////////////////////



void CAutomaticDlg::OnChangeSummaryFileNameEdit()
{
    // TODO: If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialog::OnInitDialog()
    // function to send the EM_SETEVENTMASK message to the control
    // with the ENM_CHANGE flag ORed into the lParam mask.

    // TODO: Add your control notification handler code here

////////////////////////////////////////////
// MY CODE STARTS HERE
//////

    // Update the variables of the controls.
    UpdateData(TRUE);

/////
// MY CODE ENDS HERE
////////////////////////////////////////////////////
}

void CAutomaticDlg::OnChangeEventFileNameEdit()
{
    // TODO: If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialog::OnInitDialog()
    // function to send the EM_SETEVENTMASK message to the control
    // with the ENM_CHANGE flag ORed into the lParam mask.

    // TODO: Add your control notification handler code here
```

```
//////////////////////////////////////////
// MY CODE STARTS HERE
//////

    // Update the variables of the controls.
    UpdateData(TRUE);

/////
// MY CODE ENDS HERE
//////////////////////////////////////////////////
}


void CAutomaticDlg::OnChangeTotalEventFileNameEdit()
{
    // TODO: If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialog::OnInitDialog()
    // function to send the EM_SETEVENTMASK message to the control
    // with the ENM_CHANGE flag ORed into the lParam mask.

    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
//////

    // Update the variables of the controls.
    UpdateData(TRUE);

/////
// MY CODE ENDS HERE
//////////////////////////////////////////////////
}


void CAutomaticDlg::OnSaveOutputFileButton()
{
    // TODO: Add your control notification handler code here

//////////////////////////////////////////
// MY CODE STARTS HERE
//////
    void print_event_to_file(FILE *);
    void print_total_event_to_file(FILE *);

    strcpy(summary_file_name, (LPCTSTR)m_summary_file_name);
    fp_summary_file = fopen(summary_file_name, "w");
        if(fp_summary_file == NULL)
            {
                MessageBox(" Error in saving summary file, check the file name! ");
                return;
            }


    strcpy(event_file_name, (LPCTSTR)m_event_file_name);
    fp_event_file = fopen(event_file_name, "w");
        if(fp_event_file == NULL)
            {
                MessageBox(" Error in saving event file, check the file name! ");
                return;
            }



    strcpy(total_event_file_name, (LPCTSTR)m_total_event_file_name);
    fp_total_event_file = fopen(total_event_file_name, "w");
        if(fp_total_event_file == NULL)
            {
                MessageBox(" Error in saving total file, check the file name! ");
                return;
            }



    fprintf(fp_summary_file, "File: %s\n", read_file_name);

    fprintf(fp_summary_file, "Number of Data Section: %d\n", file_data_section_no);

    fprintf(fp_summary_file, "\n");



    fprintf(fp_summary_file, "Detection Begining Section: %d\n",
                                                    begin_detect_data_section_no);

    fprintf(fp_summary_file, "Detection Ending Section: %d\n",
                                                    final_detected_data_section_no);
```

```
    fprintf(fp_summary_file, "Number of Data Section for Detection: %d\n",
                                                    no_of_data_section_for_detection);

    fprintf(fp_summary_file, "Total Time for Detection: %.2f s\n",
                                                    total_time_for_detection);

    fprintf(fp_summary_file, "\n");



    fprintf(fp_summary_file, "Number of Total Event: %d\n", total_event_no);

    fprintf(fp_summary_file, "Total Event Frequency: %f Hz\n", total_event_frequency);

    fprintf(fp_summary_file, "\n");


    fprintf(fp_summary_file, "Number of Written Event: %d\n", written_event_no);

    fprintf(fp_summary_file, "Number of Overlapping Event: %d\n", overlapping_event_no
);

    fprintf(fp_summary_file, "Percentage of Overlapping Event: %5.2f%%\n",
                                    overlapping_event_proportion * 100);

    fprintf(fp_summary_file, "\n");



    fprintf(fp_summary_file, "Detection Parameters:\n");

    fprintf(fp_summary_file, "Rising Threshold Slope: %f nA/s\n",
                                    global_rising_threshold_slope);

    fprintf(fp_summary_file, "Number of Points must exceed Rising Threshold Slope: %d\
n",
                                    global_rising_threshold_point_no);

    fprintf(fp_summary_file, "Threshold Amplitude: %f nA\n",
                                    global_threshold_amplitude);

    fprintf(fp_summary_file, "Decay Threshold Slope: %f nA/s\n",
                                    global_falling_threshold_slope);

    fprintf(fp_summary_file, "Proportion must exceed Decay Threshold Slope: %f\n",
                                    global_falling_time_proportion);

    fprintf(fp_summary_file, "\n");



    fprintf(fp_summary_file, "Factor of Amplitude decaying to: %.3f\n",
                                    global_decay_time);

    fprintf(fp_summary_file, "\n");
    fprintf(fp_summary_file, "\n");


    fprintf(fp_summary_file, "For Total Event File:\n");

///////print total event//////12345678123456781234567812345678123456781234567812345678
123456781234567812345678
    fprintf(fp_summary_file, "  Ampli- Overlap Overlap  Inter-    Data    Time\n");
    fprintf(fp_summary_file, "    tude  Previ-    Next   event Section       \n");
    fprintf(fp_summary_file, "             ous   Event Interv-               \n");
    fprintf(fp_summary_file, "           Event              al               \n");
    fprintf(fp_summary_file, "    (pA)                   (ms)           (ms)\n");

    fprintf(fp_summary_file, "\n");
    fprintf(fp_summary_file, "\n");


    fprintf(fp_summary_file, "For  Event File:\n");

////////////print event///////12345678123456781234567812345678123456781234567812345678
123456781234567812345678
    fprintf(fp_summary_file, "  Ampli-    Rise    Decay    Decay    Mean    Data    Time
\n");
    fprintf(fp_summary_file, "    tude    Time    Time    Time  Square Section
```

```
\n");
    fprintf(fp_summary_file, "                          Constant   Error
\n");
    fprintf(fp_summary_file, "    (pA)    (ms)    (ms)    (ms)    (pA)2            (ms)
\n");

    print_event_to_file(fp_summary_file);
    print_event_to_file(fp_event_file);


    print_total_event_to_file(fp_total_event_file);

    fclose(fp_summary_file);
    fclose(fp_event_file);
    fclose(fp_total_event_file);

/////
// MY CODE ENDS HERE
////////////////////////////////////////////////
}


//////////////////////////////////////////
// MY CODE STARTS HERE
////////
void print_event_to_file(FILE *fp_output)
{


    int counter;

    for(counter = 0;
        counter <= (written_event_no - 1);
        counter++)
        {
            fprintf(fp_output, "%8.1f", event_array[counter].amplitude * 1000);

            fprintf(fp_output, "%8.1f", event_array[counter].rise_time * 1000);

            fprintf(fp_output, "%8.1f", event_array[counter].decay_time * 1000);

            fprintf(fp_output, "%8.1f", event_array[counter].decay_time_constant * 100
0);

            fprintf(fp_output, "%8.1f", event_array[counter].mean_decay_current_error_
2
                                                        * 1000 * 1000);

            fprintf(fp_output, "%8d", event_array[counter].section_no);

            fprintf(fp_output, "%8.1f", event_array[counter].time * 1000);

            fprintf(fp_output, "\n");
        }

}
//////////
// MY CODE ENDS HERE
//////////////////////////////////////////


//////////////////////////////////////////
// MY CODE STARTS HERE
////////
void print_total_event_to_file(FILE *fp_output)
{


    int counter;

    for(counter = 1;
        counter <= (total_event_no - 1);
        counter++)
        {
            fprintf(fp_output, "%8.1f", total_event_array[counter].amplitude * 1000);

            fprintf(fp_output, "%8d", total_event_array[counter].
                                        overlapping_previouse_event_result);


            fprintf(fp_output, "%8d", total_event_array[counter].
                                        overlapping_foward_event_result);

            fprintf(fp_output, "%8.1f", total_event_array[counter].
                                        inter_event_interval * 1000);

            fprintf(fp_output, "%8d", total_event_array[counter].section_no);
```

```
            fprintf(fp_output, "%8.1f", total_event_array[counter].time * 1000);

            fprintf(fp_output, "\n");
        }

}
//////////
// MY CODE ENDS HERE
////////////////////////////////////////////
```